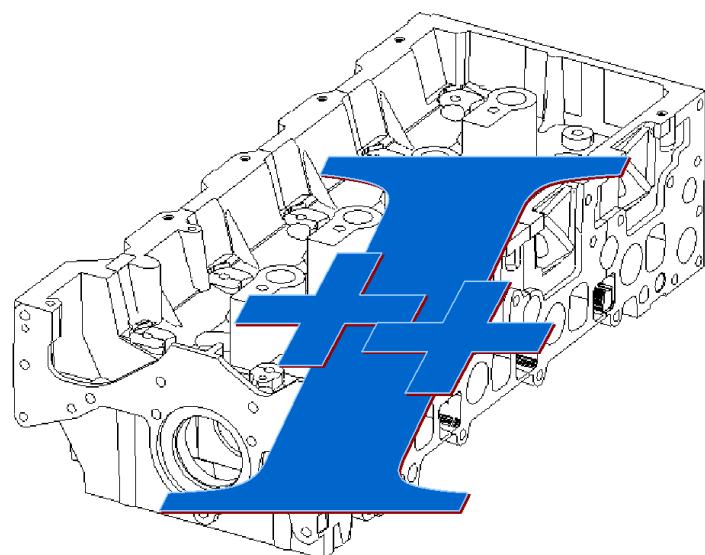




DAIMLERCHRYSLER

VOLVO



DME – Interface

Release 1.30

Contents:

1 I++ WORKING GROUP INFORMATION.....	8
1.1 This specification was created with the assistance of	8
1.2 The goal	8
1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)	8
1.4 Requirement.....	8
1.5 What is the intention of the specification ?	8
1.6 Schedule steps	10
1.7 History	10
2 PHYSICAL SYSTEM LAYOUT	11
2.1 DME-Interface Implementations	12
2.2 DME-Interface Model	12
2.3 Logical System Layout	13
2.4 DME-Interface and Subsystems	14
2.4.1 Application.....	14
2.4.2 Monitor	14
2.4.3 Diagnostics.....	14
2.4.4 Info	14
3 HIERARCHY OF COMMUNICATION	16
3.1 Layers	16
3.2 Examples of basic use cases	17
3.2.1 Sequence Diagram: StartSession, EndSession.....	17
3.2.2 Sequence Diagram: Standard Queue Communication.....	17
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events) ...	18
3.2.4 Sequence Diagram: Handling of Unsolicited Errors	19
4 EVENTS	20
4.1 Transaction events, syntax.....	20
4.2 One shot events	21
4.3 Multiple shot events.....	21
4.4 Server events	21

5 OBJECT MODEL	22
5.1 Explanation	22
5.2 Reduced Object Model.....	23
5.3 Full Object Model.....	24
5.4 Packaging for visualization.....	25
5.5 Contents of server.....	25
5.6 Contents of dme	27
5.7 Contents of cartcmm	28
5.8 Contents of cartcmmwithrotarytable	28
5.9 Contents of toolchanger	29
5.10 Contents of lib and unspecified	30
6 PROTOCOL.....	31
6.1 Communication.....	31
6.1.1 Character set.....	31
6.1.2 Units	31
6.1.3 Enumeration	31
6.1.4 Definitions used in formats	31
6.2 Protocol Basics	33
6.2.1 Tags.....	33
6.2.2 General line layout	34
6.2.2.1 CommandLine	34
6.2.2.2 ResponseLine	34
6.2.2.3 Definitions	35
6.2.3 Transactions	35
6.2.3.1 Example.....	35
6.2.4 Events.....	36
6.2.4.1 Examples	36
6.2.5 Errors.....	37
6.3 Method Syntax	38
6.3.1 Server Methods	38
6.3.1.1 StartSession()	38
6.3.1.2 EndSession().....	38
6.3.1.3 StopDaemon(..)	38
6.3.1.4 StopAllDaemons().....	39
6.3.1.5 AbortE()	39
6.3.1.6 GetErrorInfo()	40
6.3.1.7 ClearAllErrors()	40
6.3.1.8 Information for handling properties	42
6.3.1.9 GetProp(..)	42

6.3.1.10	GetPropE(..)	42
6.3.1.11	SetProp(..)	42
6.3.1.12	EnumProp(..)	42
6.3.1.13	EnumAllProp(..)	43
6.3.2	DME Methods	44
6.3.2.1	Home()	44
6.3.2.2	IsHomed()	44
6.3.2.3	EnableUser()	44
6.3.2.4	DisableUser()	44
6.3.2.5	IsUserEnabled()	45
6.3.2.6	OnPtMeasReport(..)	45
6.3.2.7	OnMoveReportE(..)	45
6.3.2.8	GetMachineClass()	45
6.3.2.9	GetErrStatusE()	46
6.3.2.10	GetXtdErrStatus()	46
6.3.2.11	Get(..)	46
6.3.2.12	GoTo(..)	47
6.3.2.13	PtMeas(..), PtMeasIJK(..)	47
6.3.2.14	Information for Tool Handling	49
6.3.2.15	Tool()	49
6.3.2.16	FindTool(..)	49
6.3.2.17	FoundTool()	50
6.3.2.18	ChangeTool(..)	50
6.3.2.19	SetTool(..)	50
6.3.2.20	AlignTool(..)	50
6.3.2.21	GoToPar()	51
6.3.2.22	PtMeasPar()	51
6.3.2.23	EnumTools()	51
6.3.3	CartCMM Methods	53
6.3.3.1	SetCoordSystem(..)	54
6.3.3.2	GetCoordSystem()	54
6.3.3.3	GetCsyTransformation(..)	54
6.3.3.4	SetCsyTransformation(..)	55
6.3.3.5	X()	55
6.3.3.6	Y()	55
6.3.3.7	Z()	55
6.3.3.8	IJK()	56
6.3.3.9	X(..)	56
6.3.3.10	Y(..)	56
6.3.3.11	Z(..)	56
6.3.3.12	IJK(..)	57
6.3.3.13	R()	57
6.3.4	ToolChanger Methods	58
6.3.5	Tool Methods (Instance of class KTool)	58
6.3.5.1	GoToPar()	58
6.3.5.2	PtMeasPar()	58
6.3.5.3	ReQualify()	58
6.3.6	GoToPar Block	58
6.3.7	PtMeasPar Block	59
6.3.8	A(), B(), C()	60
6.3.9	A(..), B(..), C(..)	60

7 ADDITIONAL DIALOG EXAMPLES	61
7.1 StartSession	61
7.2 Move 1 axis	61
7.3 Probe 1 axis	61
7.4 Move more axes in workpiece coordinate system.....	62
7.5 Probe with more axis.....	62
7.6 Set property.....	62
7.7 Get, read property	63
8 ERROR HANDLING	65
8.1 Classification of Errors	65
8.2 List of I++ predefined errors	65
9 MISCELLANEOUS INFORMATION.....	67
9.1 Coordination of company related extensions.....	67
9.2 Initialization of TCP/IP protocol-stack	67
9.3 Closing TCP/IP connection.....	67
9.4 EndSession and StartSession	67
9.5 Pre-defined Server events	67
9.5.1 KeyPress	67
9.5.2 Clearance or intermediate point set.....	68
9.5.3 Pick manual point	68
9.5.4 Change Tool request	68
9.5.5 Set property request	68
9.5.6 Additional defined keys	68
9.6 Reading part temperature	68
9.7 Links to important sites	69
10 MULTIPLE ARM SUPPORT.....	70
11 SCANNING.....	71
11.1 Preliminaries.....	71
11.1.1 Hints:	71
11.1.2 OnScanReport.....	71

11.2 Scanning known contour	72
11.2.1 ScanOnCircleHint.....	72
11.2.2 ScanOnCircle.....	72
11.2.3 ScanOnLineHint	73
11.2.4 ScanOnLine	73
11.3 Scan unknown contour	75
11.3.1 ScanUnknownHint.....	75
11.3.2 ScanInPlaneEndIsSphere	75
11.3.3 ScanInPlaneEndIsPlane	76
11.3.4 ScanInPlaneEndIsCyl	77
11.3.5 ScanInCylEndIsSphere	79
11.3.6 ScanInCylEndIsPlane	80
11.4 Scanning Examples	82
11.4.1 Scanning known contour circle	82
11.4.2 Scanning unknown contour	82
12 ROTARY TABLE.....	84
12.1 AlignPart()	84
APPENDIX A C++ AND HEADER FILES FOR EXPLANATION.....	85
A.1 \main\main.cpp.....	85
A.2 \server.....	85
A.2.1 \server\server.h	85
A.2.2 \server\part.h.....	86
A.2.3 \server\server.cpp	86
A.3 \dme.....	88
A.3.1 \dem\dme.h	88
A.4 \cartcmm.....	89
A.4.1 \cartcmm\cartcmm.h.....	89
A.4.2 \cartcmm\eulerw.cpp	90
A.5 \cartcmmwithrottbl.....	91
A.5.1 \cartcmmwithrottbl\cartcmmwithrottbl.h	91
A.6 \toolchanger	91
A.6.1 \toolchanger\toolchanger.h	91
A.6.2 \toolchanger\tool.h	93
A.6.3 \toolchanger\toolab.h	94
A.6.4 \toolchanger\toolabc.h	94
A.6.5 \toolchanger\gotoparams.h	95
A.6.6 \toolchanger\ptmeaspars.h	96
A.6.7 \toolchanger\param.h	96
A.7 Most important of lib.....	97
A.7.1 \lib\axis.h	97

A.7.2	\lib\ eulerw.h.....	98
A.7.3	\lib\ tag.h.....	98
A.7.4	\lib\ ipptypedef.h	99
A.7.5	\lib\ ippbaseclasses.h	99

1 I++ Working Group Information

1.1 This specification was created with the assistance of

Hans-Martin Biedenbach,	AUDI AG
Josef Brunner,	BMW
Kai Gläsner,	DaimlerChrysler
Dr. Günter Moritz,	Messtechnik Wetzlar
Jörg Pfeifle,	DaimlerChrysler
Josef Resch,	Zeiss IMT

I++ is a working group of five European Car manufacturers (Audi, BMW, DaimlerChrysler, VW and Volvo).

1.2 The goal

The I++ working group defined a requirement specification with the goal to achieve a new programming system for inspection devices. (not only for CMM's)

This specification will describe the I++ application protocol for the following types of DME's:

- 3D coordinate measuring machines including multiple carriage mode
- Form testers
- Camshaft, crankshaft measuring machines

The spec is created to have a common interface to give the possibility to connect different application packages to all DMEs.

1.3 Sub Working group I++ DME Interface (Dimensional Measuring Equipment)

I++ turn one's attention to the difficulties of the interfaces. So I++ defined a team, who are responsible to work out a requirement specification for a neutral I++ DME interface.

1.4 Requirement

We demand a clear definition, that the DME vendor is responsible for the accuracy of his measurement equipment, in the sense that all necessary functions related to the equipment accuracy have to be implemented in the neutral I++ DME interface.

All calibration data, no matter where created, must be stored in the DME interface.

NIST will produce tools for testing the I++ DME Interface specification. These would be made freely available outside NIST. Simulated Server/Client for verification, development and certification scenarios will be provided.

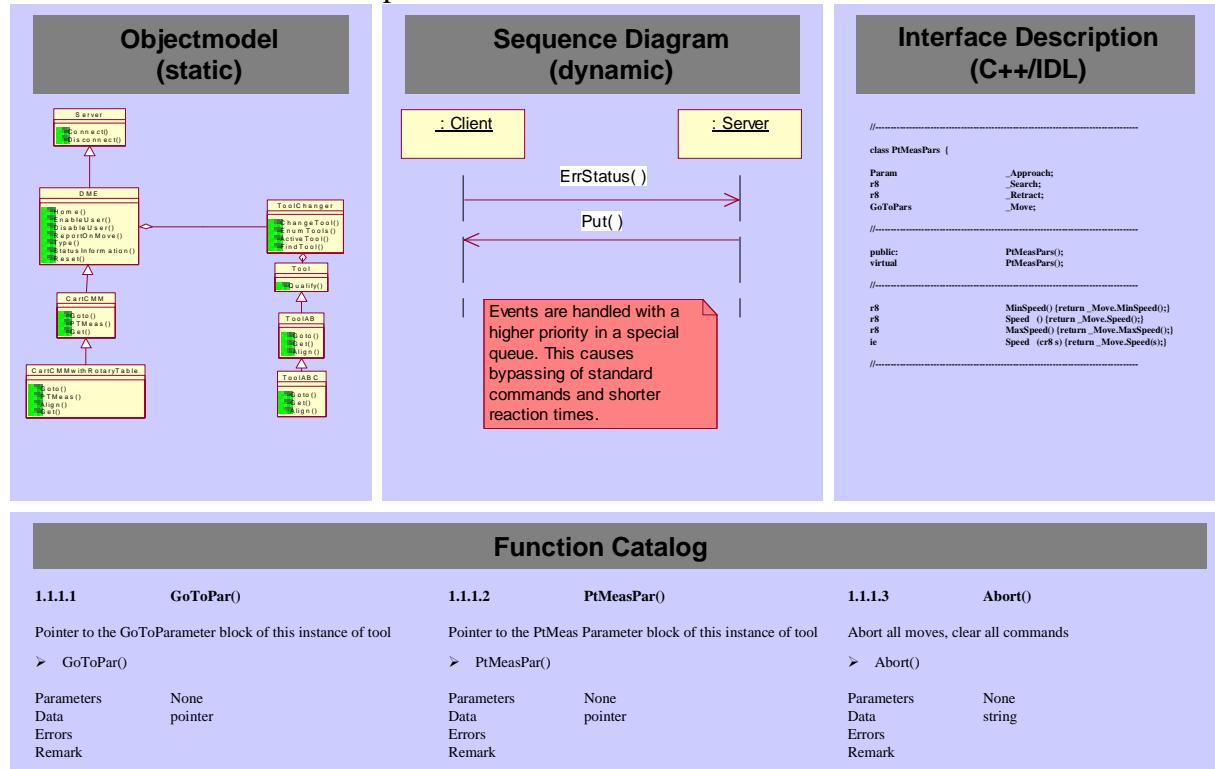
1.5 What is the intention of the specification ?

- To use State-Of-The-Art technology but useable in legacy systems
Definition of interface should be independent

of transport-layer and transport-technology

- It should provide Scaleability
“an easy machine should have an easy interface”
- Extendability
It should be possible to add new types of machines
- Encapsulation
The complexity and vendor-specific know-how of the real machine should, can be hidden behind the interface
- Self-Explaining, Consistent, Complete
Though being complex the interface should be in a notation that can be easily understood

Picture 1: Methods for description



The following requirement specification is capable of further development. This means the specification is valid for CMM's as well as other measurement equipment.

1.6 Schedule steps

Changes from 1.0 to 1.1 :	Multiple arms (port numbers...)
Changes from 1.1 to 1.2 :	Scanning, hints, collision handling
Changes from 1.2 to 1.3 :	Rotary table Note: Versions 1.2 and 1.3 have been merged to 1.3!
Changes from 1.3 to 1.4 :	Form testers
Changes from 1.4 to 1.5 :	Camshaft, crankshaft measuring machines
Changes from 1.5 to 2.0 :	Optical sensors (based on OSIS requirements)

Unscheduled extension:

- Probe-calibration-parameters-protocol
 - Separate GUI and qualification routines, handle qualification process in client application,
 - List of input-parameters necessary for calibration, all calibration data, no matter where created, must be stored in the DME, for simple probes e.g. indexable touchtrigger-probes PH9-type.
- Add Jog-Box-Display methods
- Use Unicode for strings
- Export tool-assembly information
- New CsY's: JogDisplayCsY, JogMoveCsY, SensorCsY
- Handling more than one socket between client and server

1.7 History

1.1 Multiple arms:

Changes: 6.3.3, 6.3.3.3, 6.3.3.4, 10 becomes Appendix A

Added: 10

1.3 Scanning:

Improvements: 6.1.1, 6.3.3

Added: 11

1.3 Rotary Table and Various:

Improvements: 1.6, 1.7, 2., 6.1.4, 6.2.1, 6.2.3.1, 6.3.1.7, 6.3.2.8, 6.3.2.11, 6.3.2.13, 6.3.6, 6.3.7, 7.7, 8.1, 8.2, 9.1, 9.5.1

Added: 6.2.3.13, 6.2.8, 6.2.9, 6.3.2.23, 6.3.3.13, 9.5.6, 9.6, 9.7, 12

2 Physical System Layout

This section is intended to help explain the context of this specification. It is not part of the specification.

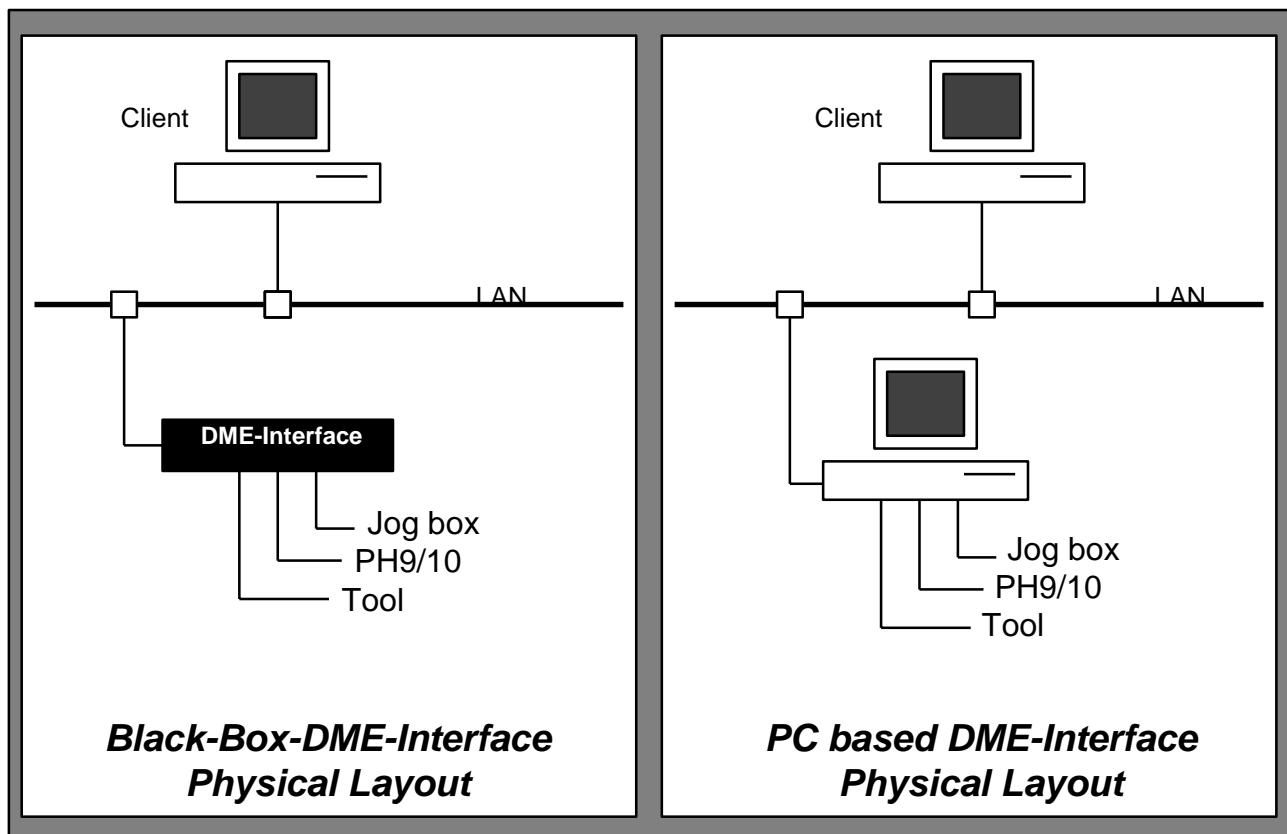
Picture 2 shows two examples of the physical system layout for these types of machines.

In both examples the main components are

Client computer (Client) and

DME-Interface

Machine (including frame, motors, scales, ...)



Client and DME-Interface are connected through a local area network (LAN).

Both client and DME-Interface use TCP/IP sockets for communication.

The client computer runs the application software for the measurement task.

The DME-Interface implements all functionality required to drive the machine.

The application software on the client talks to the DME-Interface in order to execute elementary measurement tasks (picking points, scanning, ...).

This specification describes the protocol that the client uses to run the machine through the DME-Interface.

Explanations: In the following lines client is used synonym for the application software, server is used for DME. Client and server can be on different computers, but they can also run on the same hardware being connected by TCP/IP socket.

2.1 DME-Interface Implementations

The main difference between the two implementations of the DME-Interface in Picture 1 is the physical implementation of the DME-Interface, which is

- PC based or
- “Black Box” based

While the PC based DME-Interface provides a direct physical (screen, keyboard) user interface the black box based system provides no direct user interface, PC based systems may provide additional low-level user interfaces that help the user to control and monitor the machine.

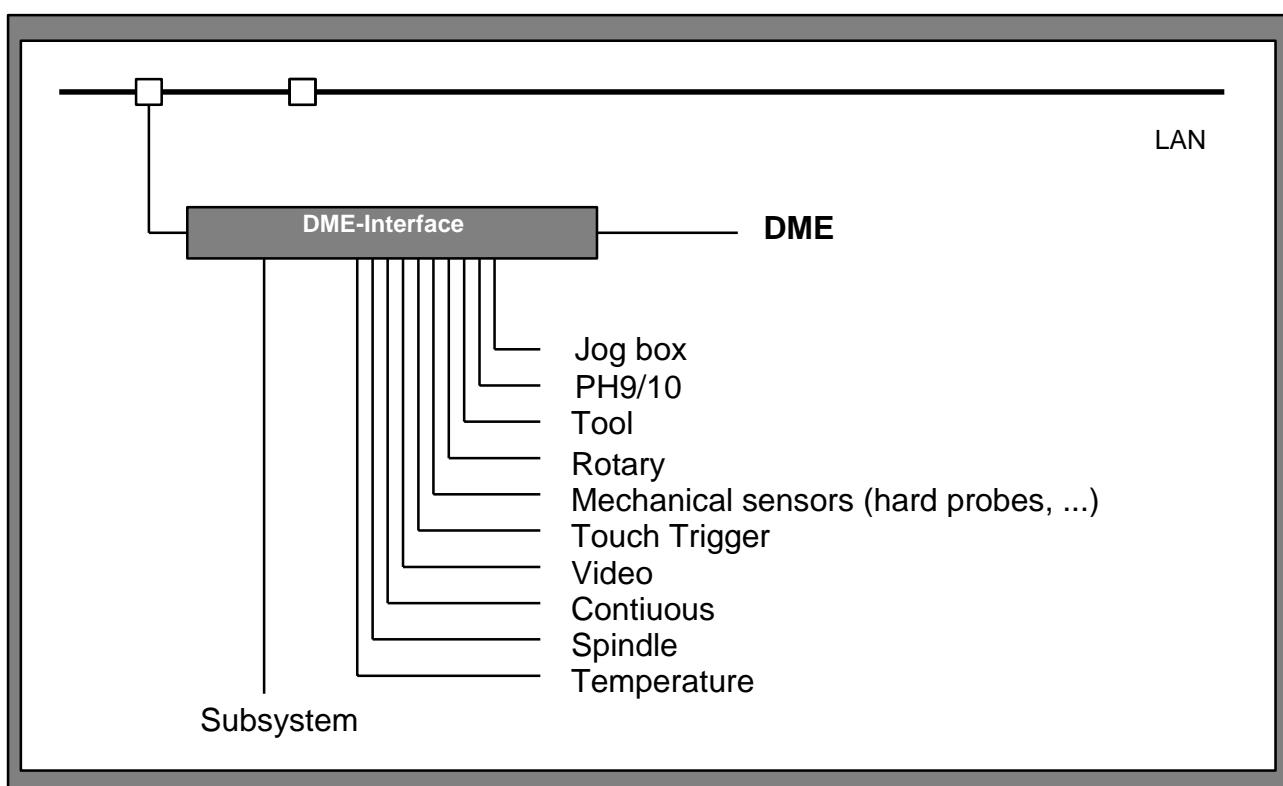
“Black Box” based systems have a potential cost advantage.

2.2 DME-Interface Model

Picture 3 shows the system layout we will use in this document for explanations.

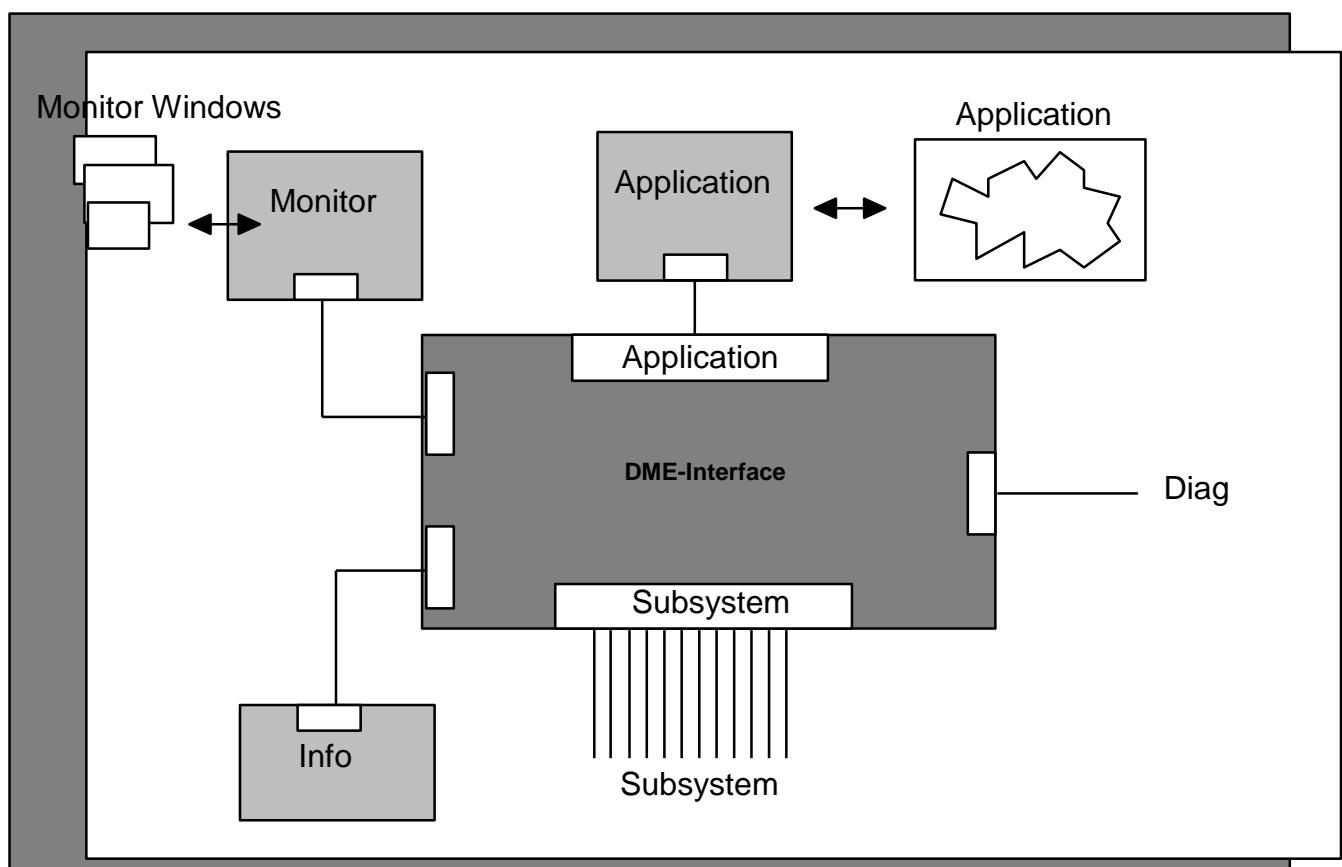
It is important to recognize that all subsystems are linked to the DME-Interface.

This implies that the client must use the protocol to access subsystem functionalities, like rotating a PH10.



2.3 Logical System Layout

Picture 4 shows the logical layout of the system with the following components:
DME-Interface and subsystems
Application
Monitor
Diagnostics



2.4 DME-Interface and Subsystems

The DME-Interface is a piece of software that runs on a PC based or "black box" based piece of hardware. This hardware connects to all subsystems (Picture 3).

The DME-Interface handles all subsystems and provides TCP/IP sockets for communication. When the hardware is powered up and the DME-Interface is started, the DME-Interface will create up to 4 TCP/IP ports:

Application port	(required)	port No. 1294
Monitor port	(optional)	
Diagnostics port	(optional)	
Info port		(optional in V.1.0 will be required in future version)

2.4.1 Application

The application is a piece of software that runs on the client computer and that uses the application port to run the DME.

This specification describes the protocol used on the application port.

The port number 1294 is internationally defined for this connection.

This port is the only one to start any movements of machine or tool. Only this allows changing any parameter.

2.4.2 Monitor

The machine monitor (monitor) is a piece of software that is used to display controller specific information like current machine position, active probe, ...

It connects to the monitor port to receive the displayed information from the DME-Interface. The monitor is an optional component.

The controller may implement an equivalent functionality, for example by displaying the machine position on the jog box display.

In most cases the DME vendor will supply the monitor.

A description of the monitor is not part of this specification.

2.4.3 Diagnostics

The machine diagnostics (diagnostics) is a piece of software that is used to display diagnostic information necessary to service, repair or set up the DME.

It connects to the diagnostic port to receive information from the DME-Interface.

The diagnostic is an optional component.

The DME vendor supplies the diagnostics.

A description of the diagnostics is not part of this specification.

2.4.4 Info

The info is a piece of software that runs on a client computer. The info obtains information from the DME-Interface through the info port and provides the information to the client (axis, sensors,...).

This specification describes the protocol used on the info port.

The functions possible on the info port are a subset of the functions possible on the application port. **On this port machine moving commands and setting of parameters are prohibited. Only information receiving dialog is allowed.**

3 Hierarchy of Communication

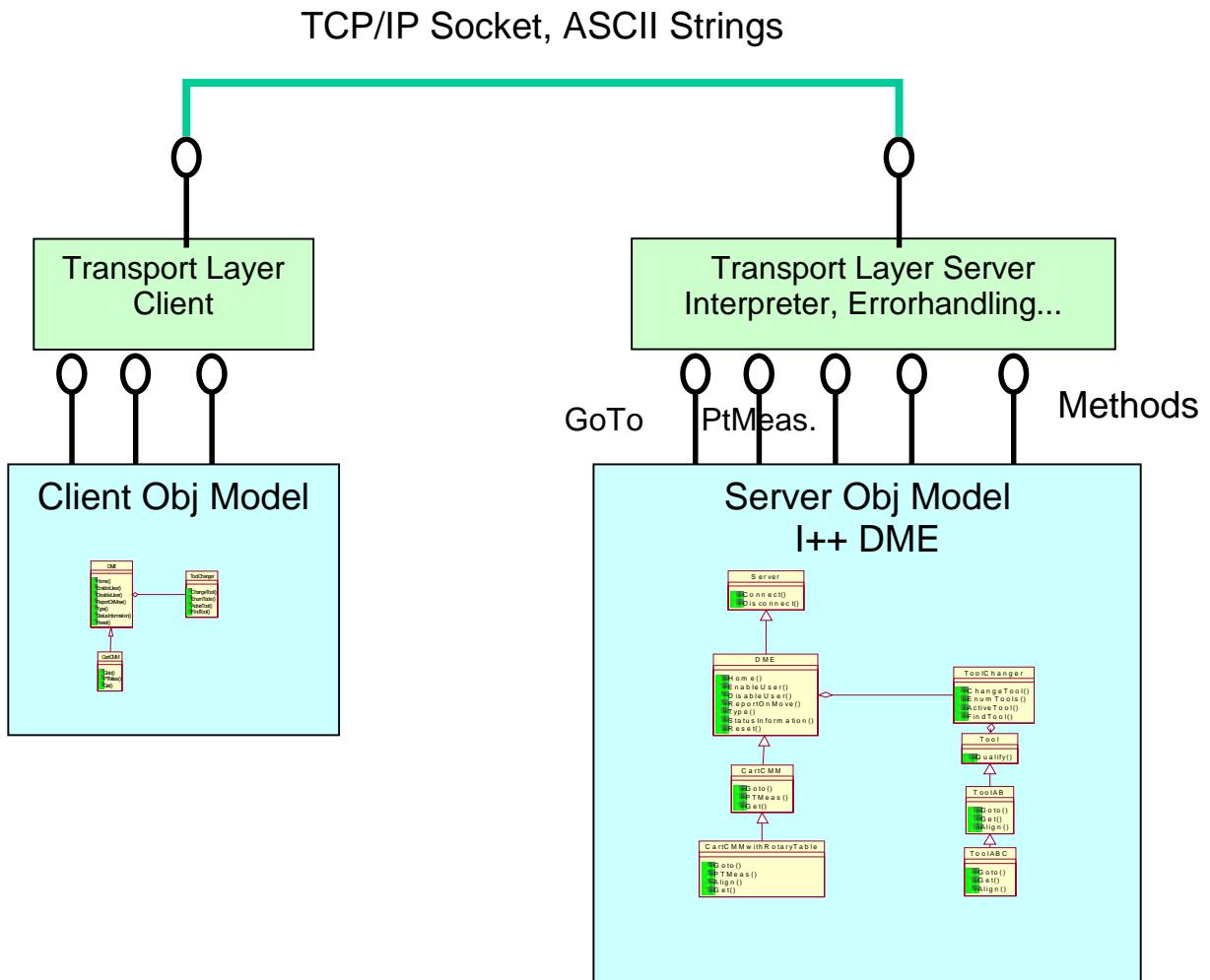
3.1 Layers

The properties of the measuring equipment and the methods to handle them are defined by the object model, see picture 13.

The actual defined transport layer is to transmit ASCII strings via TCP/IP socket.

The layers are separated to have the chance to change the transport layer to future technologies.

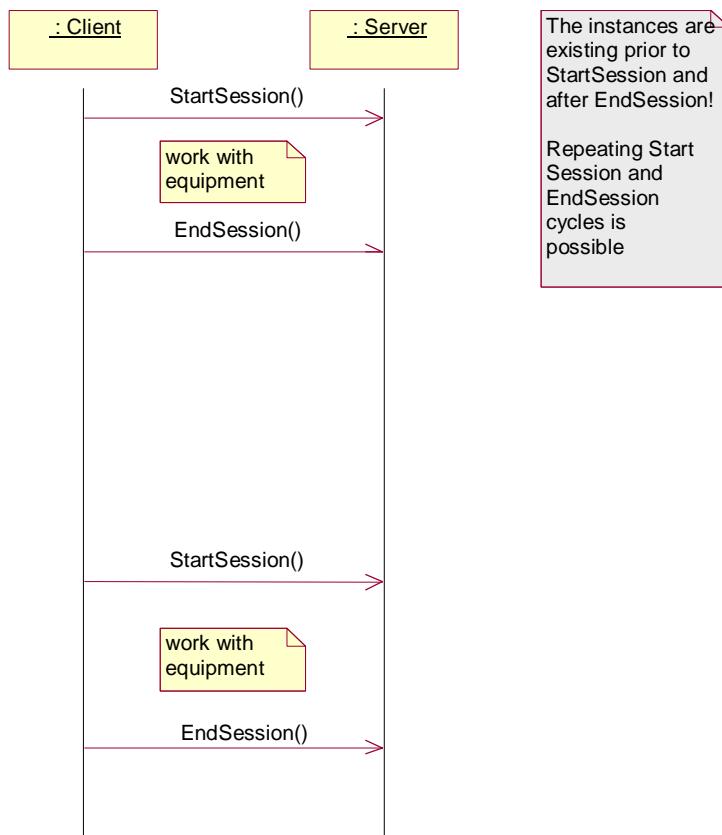
Picture 5



3.2 Examples of basic use cases

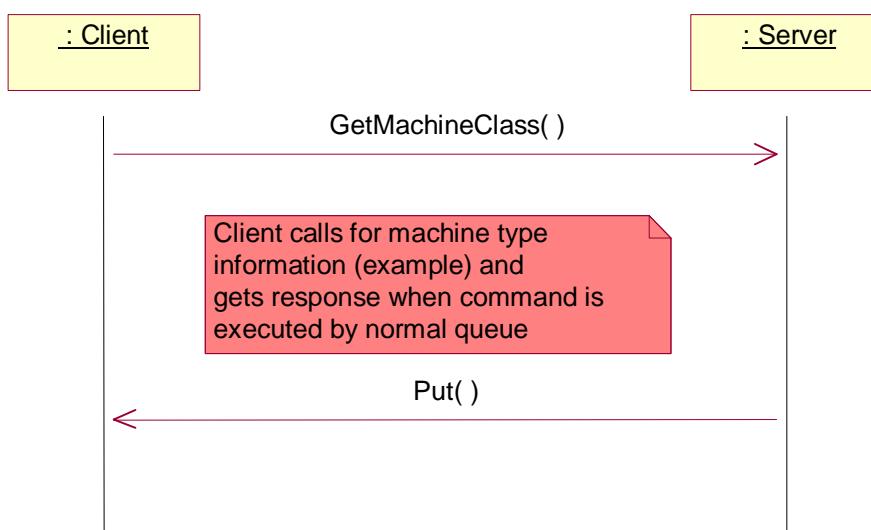
3.2.1 Sequence Diagram: StartSession, EndSession

Picture 6



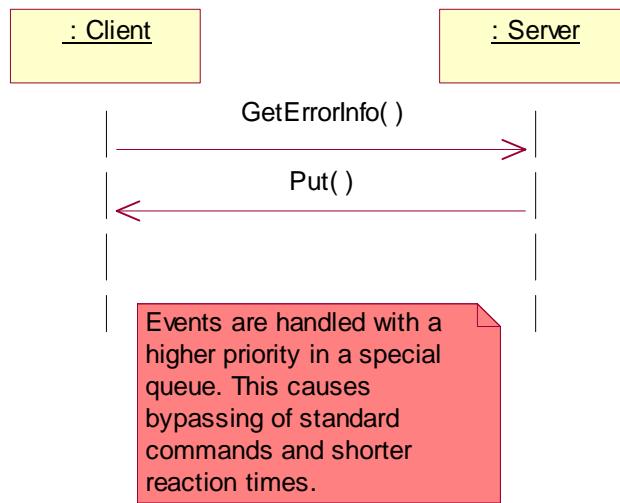
3.2.2 Sequence Diagram: Standard Queue Communication

Picture 7



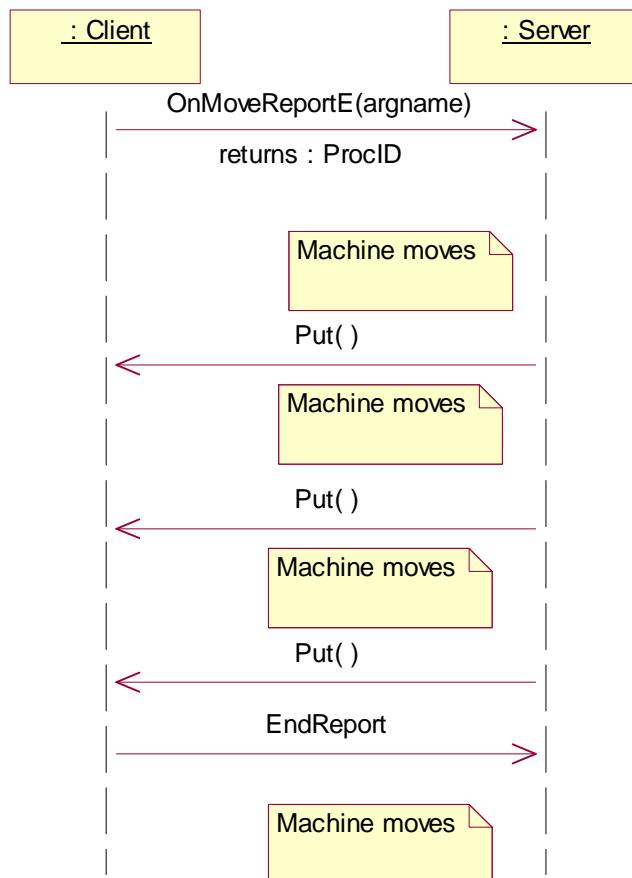
Sequence Diagram: Event, Fast Queue Communication (Single Shot Events)

Picture 8



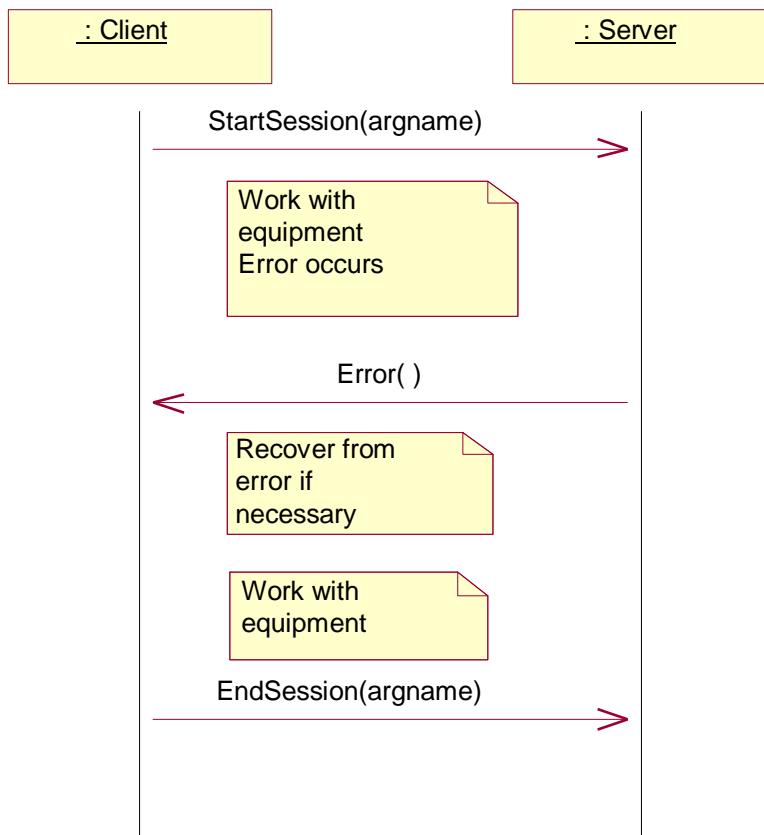
3.2.3 Sequence Diagram: Event, Fast Queue Communication (Multiple Shot Events)

Picture 9



3.2.4 Sequence Diagram: Handling of Unsolicited Errors

Picture 10

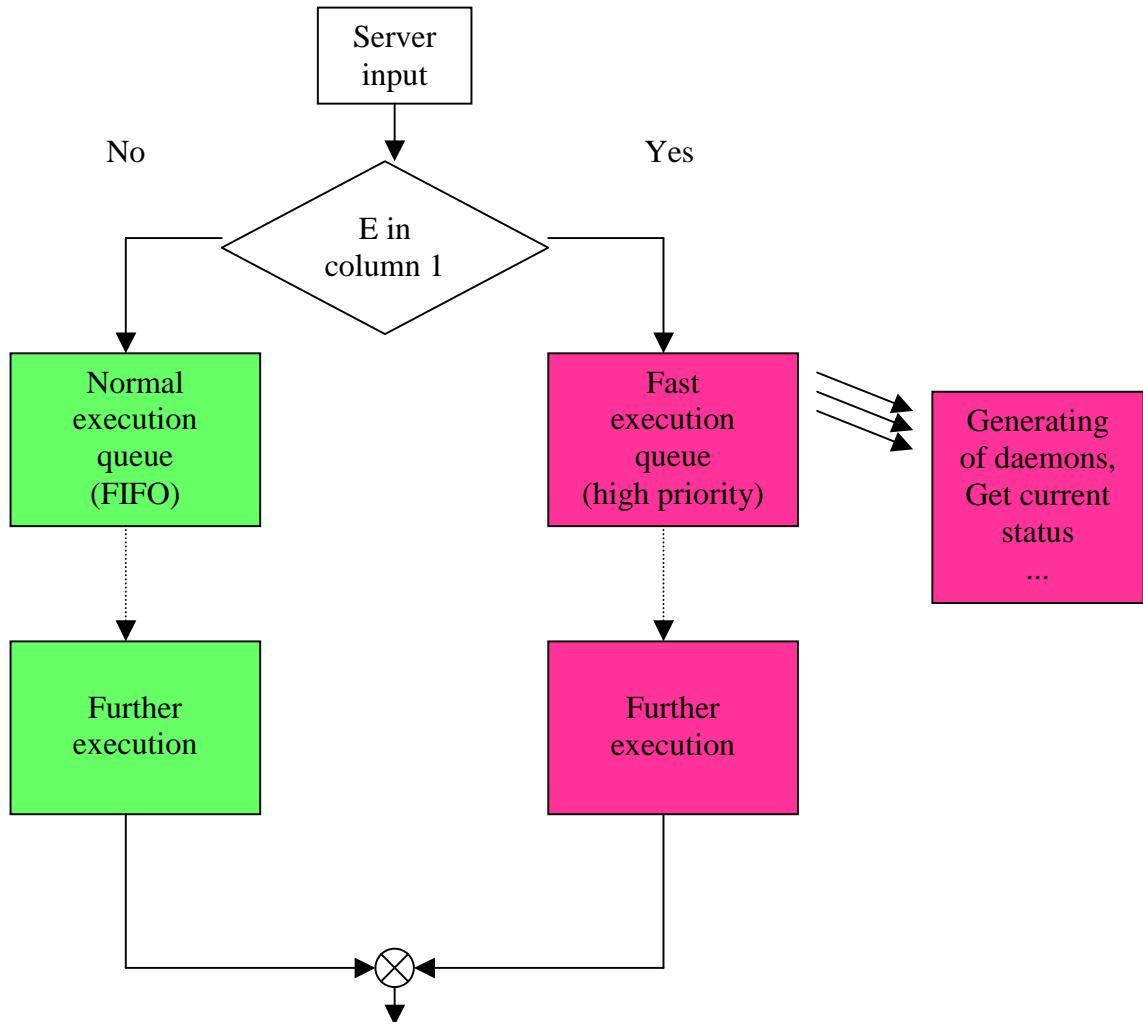


4 Events

To increase performance and to reduce traffic on the interface the Event transactions are created. Events use tags starting with E.

Picture 11: Explanation of the difference between normal and fast queue.

See also sequence diagram section 6.2.4



4.1 Transaction events, syntax

Event transactions are initiated by the client.

Event requests are handled by the server with a higher priority than the synchronous communication. This means that the requests can bypass the normal command queue in the server.

In addition to normal transaction processing, the server will trigger an event. Legal tags are tags starting with E0001 up to E9999. The tag E0000 is reserved for events with no relation to legal tags.

4.2 One shot events

These Events are used to generate exactly one asynchronous reaction of the server. F.I. getting asynchronous status or position information.

The transaction creates a daemon that triggers an event. The daemon will die after firing the event.

4.3 Multiple shot events

The transaction creates a daemon that triggers events based on a condition. The client must stop this daemon explicitly by a StopDaemon (“Event transaction tag”) method.

4.4 Server events

Server events use tag E0000. They are used to report manual hits, key strokes, supported machine status changes...

5 Object Model

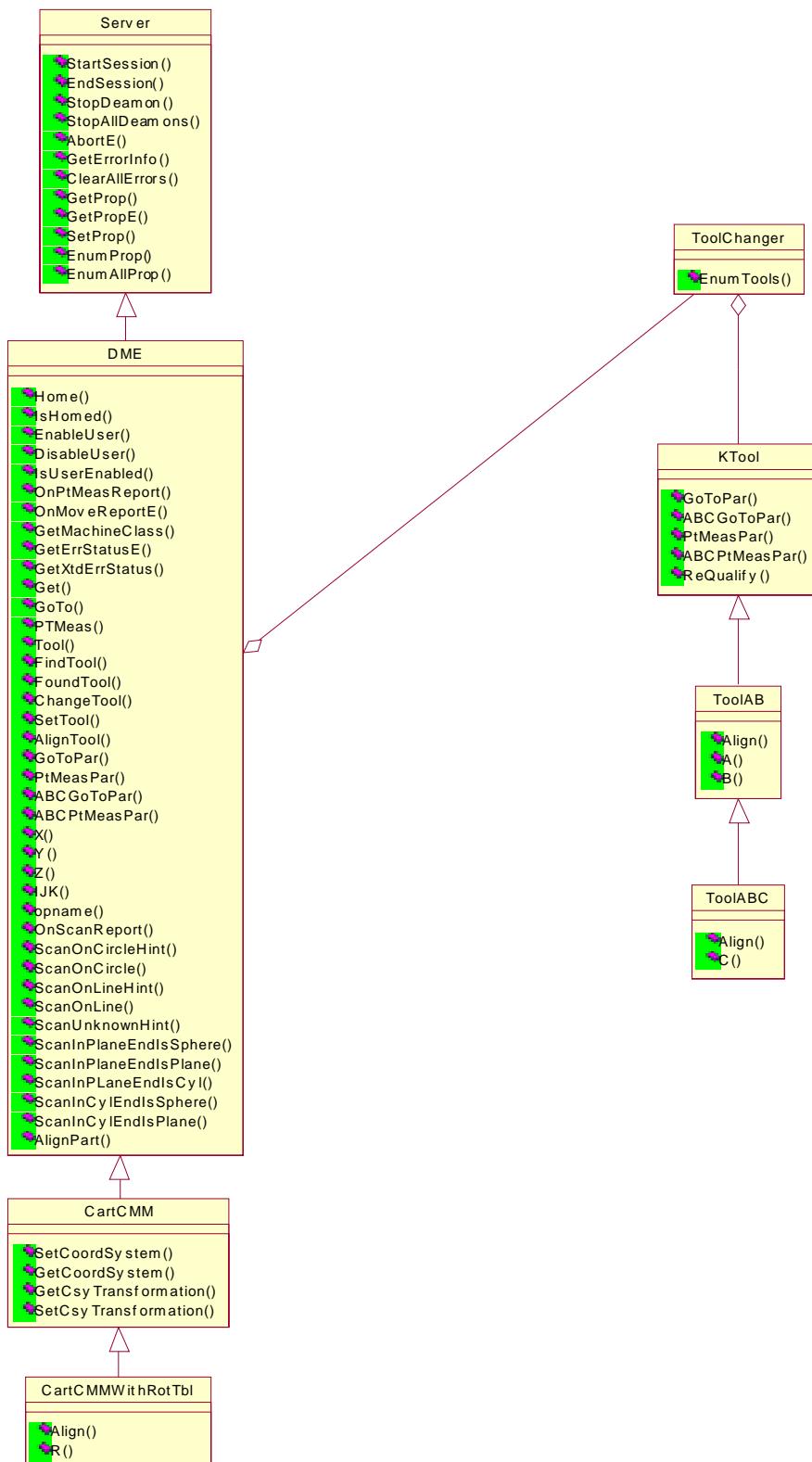
5.1 Explanation

The following diagrams (picture 12 and 13) show the designed class structure of the interface. It shows

- the classes representing the main components of real coordinate measurement equipment (in this, first case coordinate measurement machine)
- the organization of methods and properties in these classes
- the relations between the main classes, the generalizations (specialization vice versa), the aggregations...
- this object model defines the structure of the interface and the syntax. It defines how to set and get the properties of the virtual components of this machine (section 6)
- Picture 11 is generated to help at a first step with the most important commands.
- Picture 12 is reengineered from and consistent with the header files (section 9). It shows also programming aspects as virtual definitions of methods in upper classes as DME and also property aspects as GoToPar and PtMeasPar blocks.

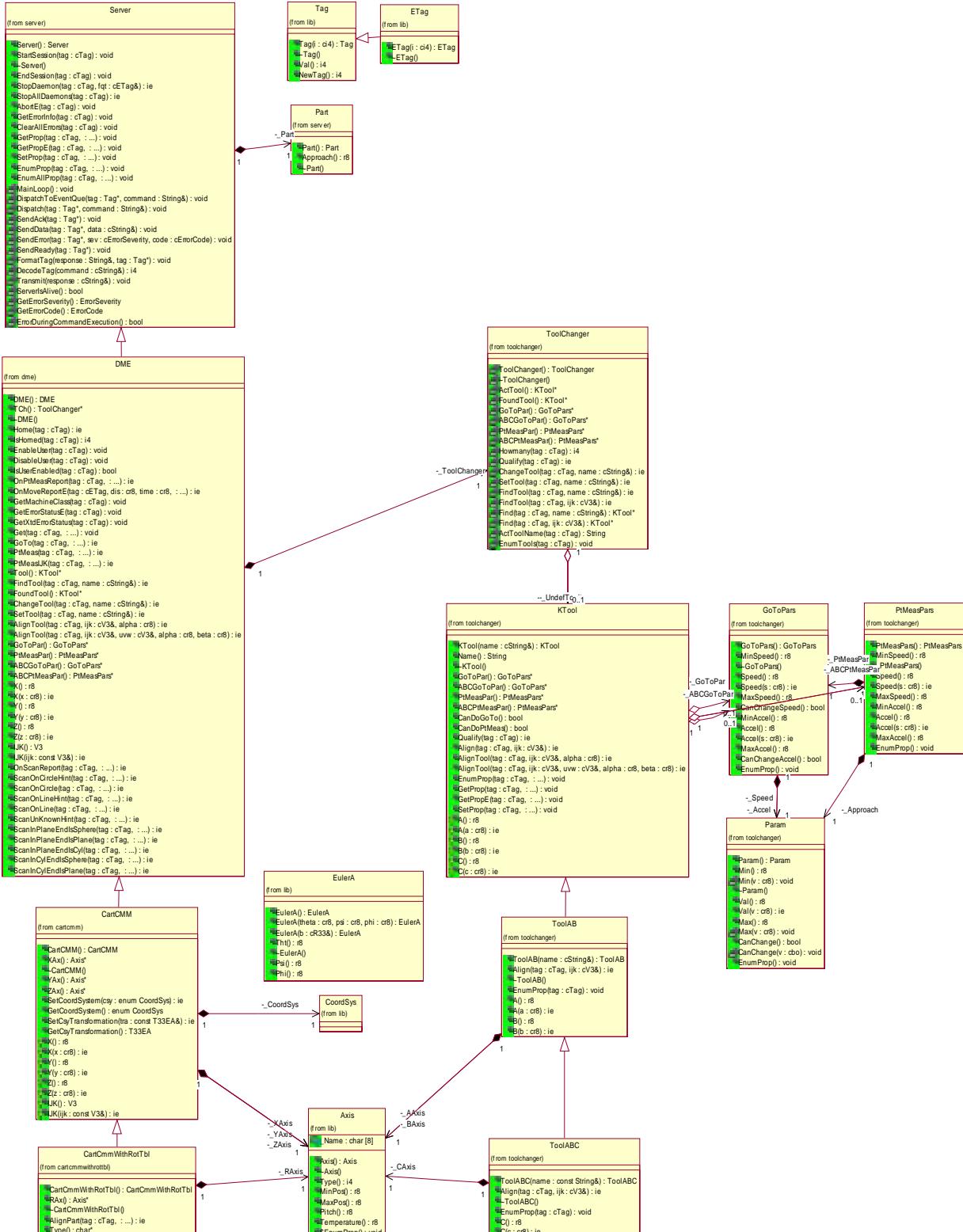
5.2 Reduced Object Model

Picture 12, basics, outside view, method oriented



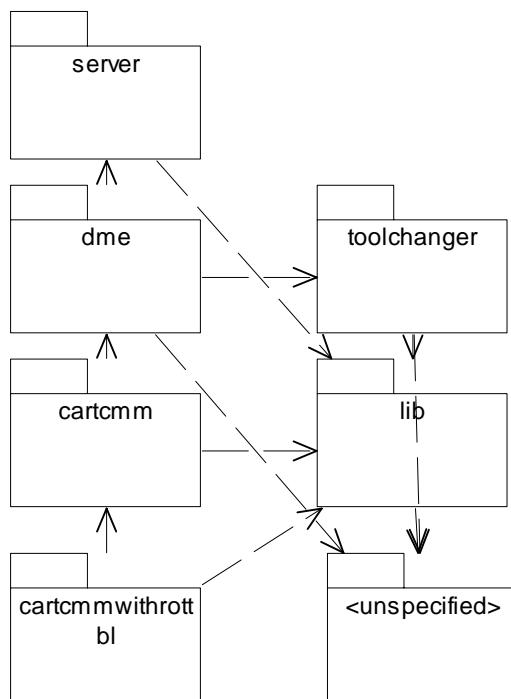
5.3 Full Object Model

Picture 13, please zoom the .pdf file view.



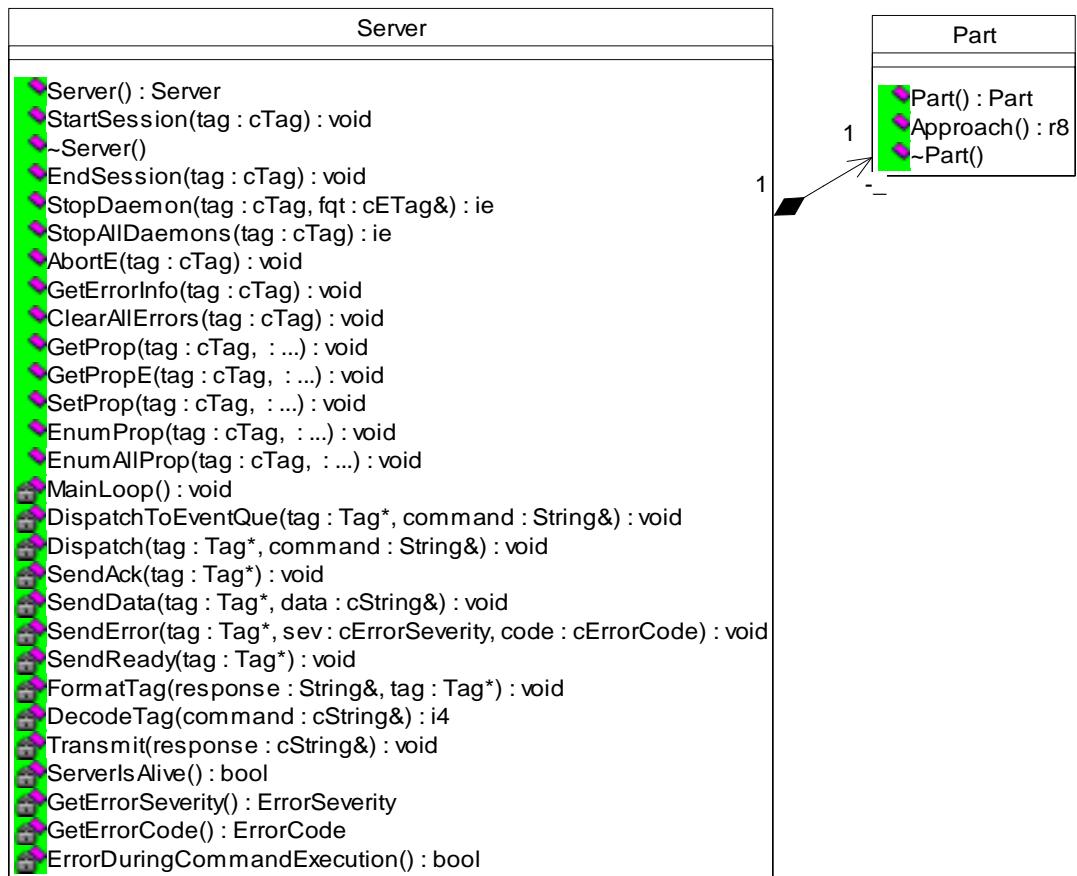
5.4 Packaging for visualization

Picture 14



5.5 Contents of server

Picture 15



5.6 Contents of dme

Picture 16

DME
<ul style="list-style-type: none">• DME() : DME• TCh() : ToolChanger*• ~DME()• Home(tag: cTag) : ie• IsHomed(tag: cTag) : i4• EnableUser(tag: cTag) : void• DisableUser(tag: cTag) : void• IsUserEnabled(tag: cTag) : bool• OnPIMeasReport(tag: cTag, ...) : ie• OnMoveReportE(tag: cETag, ds: or8, time: or8, ...) : ie• GetMachineClass(tag: cTag) : void• GetErrorStatusE(tag: cTag) : void• GetXtoErrorStatus(tag: cTag) : void• Get(tag: cTag, ...) : void• GoTo(tag: cTag, ...) : ie• PIMeas(tag: cTag, ...) : ie• PIMeasJK(tag: cTag, ...) : ie• Tod() : KTod*• FindTod(tag: cTag, name: cString8) : ie• FoundTod() : KTod*• ChangeTod(tag: cTag, name: cString8) : ie• SetTod(tag: cTag, name: cString8) : ie• AlignTod(tag: cTag, ijk: cv3&, alpha: or8) : ie• AlignTod(tag: cTag, ijk: cv3&, uwv: cv3&, alpha: or8, beta: or8) : ie• GoToPar() : GoToPars*• PIMeasPar() : PIMeasPars*• ABOGoToPar() : GoToPars*• ABOPIMeasPar() : PIMeasPars*• X() : r8• X(x: or8) : ie• Y() : r8• Y(y: or8) : ie• Z() : r8• Z(z: or8) : ie• UK() : V3• UK(ijk: const V3&) : ie• OnScanReport(tag: cTag, ...) : ie• ScanOnCircleHnt(tag: cTag, ...) : ie• ScanOnCircle(tag: cTag, ...) : ie• ScanOnLineHnt(tag: cTag, ...) : ie• ScanOnLine(tag: cTag, ...) : ie• ScanUnknownHnt(tag: cTag, ...) : ie• ScanInPlaneEndsSphere(tag: cTag, ...) : ie• ScanInPlaneEndsPlane(tag: cTag, ...) : ie• ScanInPlaneEndsCyl(tag: cTag, ...) : ie• ScanInCylEndsSphere(tag: cTag, ...) : ie• ScanInCylEndsPlane(tag: cTag, ...) : ie

5.7 Contents of cartcmm

Picture 17

CartCMM
• CartCMM() : CartCMM • XAx() : Axis* • ~CartCMM() • YAx() : Axis* • ZAx() : Axis* • SetCoordSystem(cs _y : enum CoordSys) : ie • GetCoordSystem() : enum CoordSys • SetCs _y Transformation(tr _a : const T33EA&) : ie • GetCs _y Transformation() : T33EA • X() : r8 • X(x : cr8) : ie • Y() : r8 • Y(y : cr8) : ie • Z() : r8 • Z(z : cr8) : ie • IJK() : V3 • IJK(ijk : const V3&) : ie

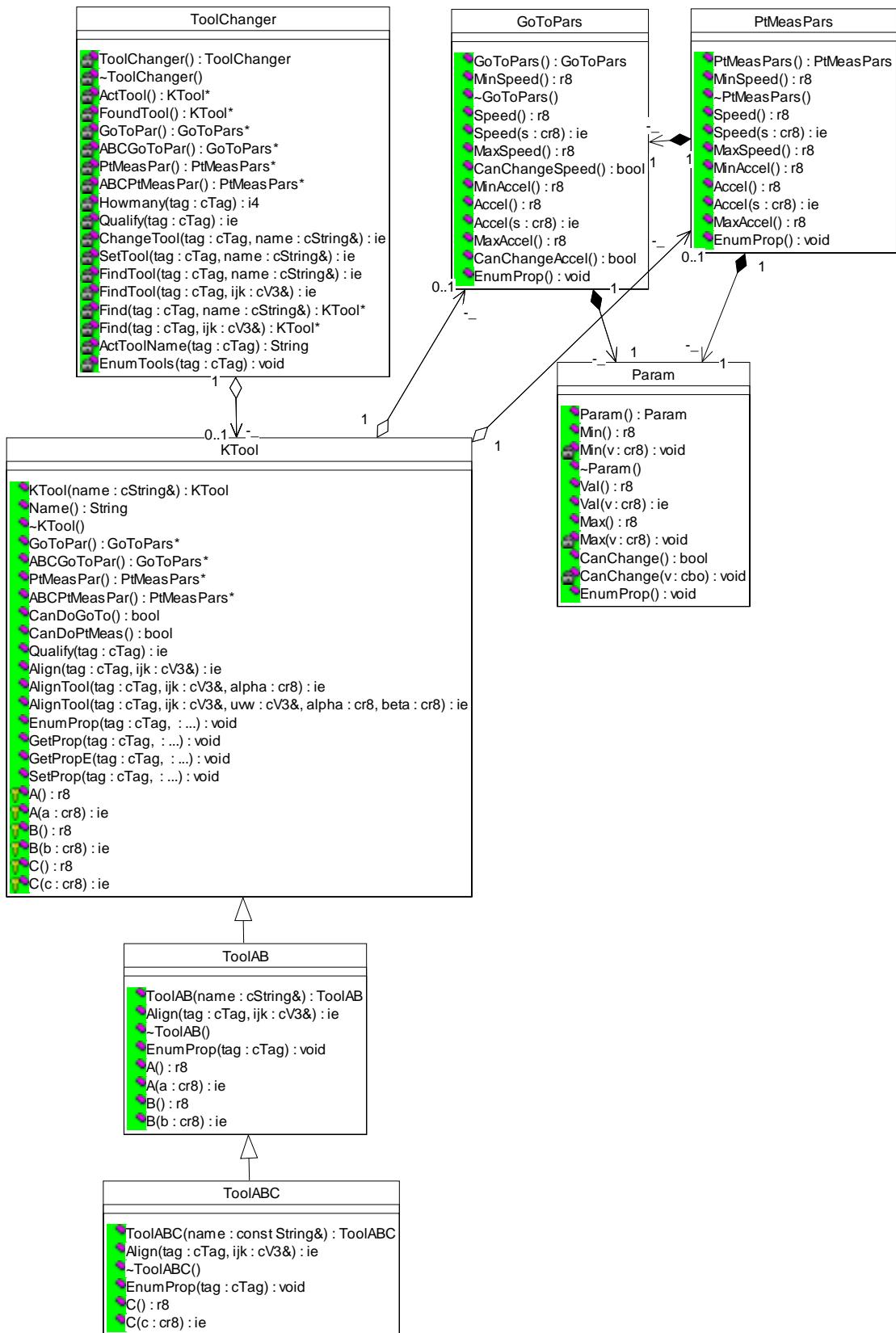
5.8 Contents of cartcmmwithrotarytable

Picture 18

CartCmmWithRotTbl
• CartCmmWithRotTbl() : CartCmmWithRotTbl • RAx() : Axis* • ~CartCmmWithRotTbl() • AlignPart(tag : cTag, : ...) : ie • Type() : char*

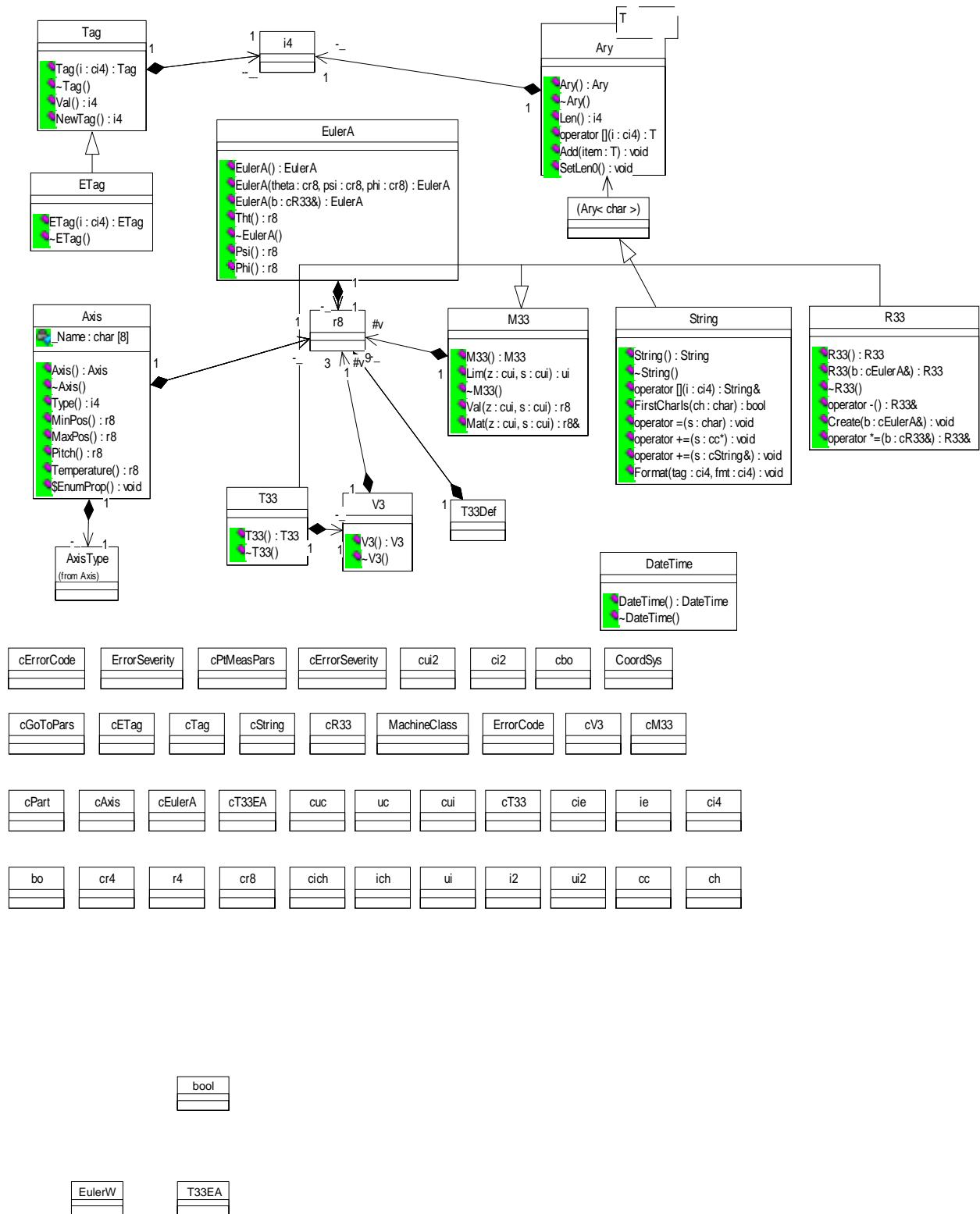
5.9 Contents of toolchanger

Picture 19



5.10 Contents of lib and unspecified

Picture 20



6 Protocol

6.1 Communication

Communication between application client and I++DME server is based on the standard TCP/IP protocol. It uses the application port with port-no. 1294.

6.1.1 Character set

All bytes received and sent on the application port of the server are interpreted as 8-bit ASCII characters. The 128's bit must always be zero.

Only characters in the range from ASCII code = 32 (space) to ASCII code = 126 (~) may be used, except that the character pair Carriage Return (<CR>, ASCII code = 13) and Line Feed (<LF>, ASCII code = 10) is used as a line terminator.

<CR> and <LF> must always be sent as a pair in the order <CR> followed by <LF>.

If the server receives a message containing any character outside of this range or containing a <CR> or <LF> anywhere except at the end of the message, the server must send an error response, using error 0007, "Illegal character".

Upper case letters and lower case letters are regarded as different letters in this protocol. In other words, the protocol is case sensitive.

6.1.2 Units

Numbers that represent measures must use the following units.

Length: millimeters

Time: seconds

Angles: decimal degrees (no minutes or seconds)

Temperature: degrees Celsius

Force: Newtons

Compound measures use combinations of these units. For example, speed (which is length per unit time) must be expressed in millimeters per second.

6.1.3 Enumeration

An enumeration is a list of zero to many items from a specified list of candidate items. Each item may appear in the enumeration list at most once, and the order in which the items appear on the enumeration list is not significant. For example, if the candidate list is (a, b, c, d, e):

1. (a, b, d) and (d, a, b) are the same enumeration, and both are legal.
2. (c, b, c) is illegal because c appears twice.
3. (c, e, h) is illegal because h does not appear in the candidate list.

6.1.4 Definitions used in formats

Char ::= “ “ !” #” ..”~” (ASCII char codes from “space” to “~” excluding “”)

String ::= “““Char {Char}”””

```

Alp      ::= "A" .. "Z" "a" .. "z" "_"
Num      ::= "0" .. "9"
BasicName ::= Alp {Alp | Num}
Name     ::= BasicName {"." BasicName}
Tag      ::= "00001" .. "99999"
ETag0    ::= "E0000"
ETag     ::= "E0001" .. "E9999"
UnsInt   ::= Num {Num}
Number   ::= ("+"|"-") Num {"."{Num}} {,,E" {"+"|"-"} Num {Num{Num}}}
Method   ::= Name {" "} "(" {" "}{ArgList{" "}})"
Property ::= Name{" "} "(" {" "}{ArgList{" "}})"
Argument ::= String | Number | Enum | Method | Property
ArgList  ::= {" "}{Arg {" "}" {" "}{Arg}}

```

6.2 Protocol Basics

- The protocol is line oriented.
- Each line must be terminated by <CR><LF>.
- The maximum number of characters in a line should not exceed 65536.
- A line sent from the client to the server is called a CommandLine.
- A line sent from the server to the client is called a ResponseLine.

In examples the terminating <CR><LF> is not shown !

The protocol is case sensitive.

6.2.1 Tags

The first 5 characters of each CommandLine represent a tag.

The client generates these tags. The client uses two types of tags:

- CommandTag
- EventTag

A CommandTag is a 5 digit decimal number with leading zeros present.

The number must be between 00001 and 09999.

Command tags are considered to be enums in the range of 00001 and 99999.

The client must make sure, that command tags send to the server are unique while the server processes the commands related to the tags. The easiest way to accomplish this is to increment the tag number each time a new command is send.

Examples of tags created by the client:

```
04711      // tag is ok
01710      // ok
00020      // ok
20          // error; only 2 digits
00000      // error; out of range must be >=00001 and <=99999
```

An EventTag is a 4 digit decimal number that is preceded by the character E(ASCII code=69).

The number must be between 0001 and 9999.

Event tags are considered to be enums in the range of E0001 and E9999.

To differ in the command layer also between the normal and fast queue, commands for the fast queue end with an upper case E. The reason is to be independent from the transport layer.

The client must make sure, that event tags send to the server are unique while the server processes the commands related to the event tags. The easiest way to accomplish this is to increment the tag number each time a new command is send.

Examples of tags created by the client:

```
E3333      // tag is ok  
E0456      // ok  
E0000      // error; out of range must be >=1 and <=9999  
E20        // error; only 3 characters  
A4711      // error; illegal first character
```

As for a CommandLine, the first 5 characters of a ResponseLine represent a tag (ResponseTag). During normal command processing by the server it will use the tag received from the client as ResponseTag so the client can use this tag to relate the ResponseLine to a CommandLine.

In addition the server can send a ResponseLine using ResponseTag E0000 for reporting unsolicited events to the client. The “Illegal tag” error message should be reported by the E0000 tag.

6.2.2 General line layout

From now on we will use

- Command as a synonym for CommandLine
- Response as a synonym for ResponseLine.

6.2.2.1 CommandLine

The first 5 characters in each CommandLine represent the CommandTag.
The character at column 6 must be a space (ASCII code = 32).
The command starts at column 7.

Command := Tag | Etag “ “ Method

6.2.2.2 ResponseLine

The first 5 characters in each Response line represent the ResponseTag.
The character at column 6 must be a space (ASCII code = 32).
The character at column 7 must be one of the following:

- &
- %
- #
- !

The meaning of the character at column 7 is explained later.
The character at column 8 must be a space when the line length is greater than 7.
Example:
00004 # X(99.93), Y(17.148)

Please note, that if not explicitly stated in a command, the order of the return data is left to the server implementation.

Therefore

00004 # Y(17.148), X(99.93)

is an equivalent response.

In addition the returned data must exactly match the requested data. No data may be omitted and no data may be added to the response line.

6.2.2.3 Definitions

In the following we will use

- Ack as a synonym for a ResponseLine where the 7th character is a &
- Transaction complete as a synonym for a ResponseLine where the 7th character is a %
- Data as a synonym for a ResponseLine where the 7th character is a #
- Error as a synonym for a ResponseLine where the 7th character is a !

6.2.3 Transactions

The basic protocol unit is a transaction. For each transaction, the client will create a tag. The tag identifies the transaction.

- Transactions are initiated by the client.
- The same tag is used during a transaction.
- Transactions can overlap, which means that a client can start a new transaction after having received an Ack from the server.
- When using overlapped transactions, tags sent to the server must be unique.
- When using overlapped transactions and the server is too busy to accept new transactions it must delay sending the Ack until it is ready to accept a new transaction.
- When using overlapped transactions, the server must make sure that the Ack, Data (Error) and Transaction complete are sent back to the client in the right order. This means, if transaction 00001 is started before transaction 00002, the server is not allowed to send a Data, Error or Transaction complete from transaction 00002 before the Transaction complete from transaction 00001. At any point in time the server is allowed to send a line starting with an EventTag.

A transaction is complete after the server sends the Transaction complete. If a transaction is complete, all processing on the driver side related to the transaction has completed.

6.2.3.1 Example

Client to Server	Server to Client	Comment
00001 Home()		Use tag 00001 for home

		command, client sends “home” command
	00001 &	Server accepts command (Ack)
	00001 %	Server reports transaction complete
00002 GoTo(X(100))		Move to x=100
	00002 &	command accepted (Ack)
	00002 %	position reached (Transaction complete)
00003 GoTo(X(100000))		moving out of limits
	00003 &	command accepted
	00003 ! Error(3, 2500, GoTo, “Machine limit encountered (Move Out Of Limits)”)	Error message
	00003 %	transaction complete
00004 Get(X(), Y())		get position of x, y axis
	00004 &	
	00004 # X(99.93), Y(17.148)	x and y position
	00004 %	Transaction complete

6.2.4 Events

At any point in time the server may notify that something happened by sending an event to the client.

If the event is triggered by a transaction, the tag used is that of the transaction.

The server must first send an Ack before it can send the Response with the EventTag.

This Response can then be sent before or after the Transaction complete.

If an event is triggered by a command, f.I. ErrStatusE, the server handles the execution of the command (responding of the error status) with a higher priority. The Transaction complete is responded in the order of the standard queue.

At any point in time the server can send a Response with EventTag E0000 to inform the client that something unsolicited has happened in the server.

6.2.4.1 Examples

Unsolicited error message

Client to Server	Server to Client	Comment
	E0000 ! Error(3, 9, HealthCheck, Emergency button activated)	An unsolicited error message occurs
		In this example the server must display error and inform user

		what to do
--	--	------------

Assume the user moves the machine using joysticks and the server wants to report this movement.

Client to Server	Server to Client	Comment
00048 EnableUser()		
	00048 &	
	00048 %	
E0553 OnMoveReportE(Time(1),Dis(20),X()Y()Z())		
	E0553 &	
	E0553 %	
		Now the user moves the machine
	E0553 # X(50), Y(433), Z(500)	
	E0553 # X(50), Y(433), Z(520)	
	...	
		Now the client wants to stop reporting of the server and sends
00049 StopDaemon(E0553)		
	00049 &	
	E0553 # X(50), Y(433), Z(530)	
	00049 %	no events with tag E0553 may follow

6.2.5 Errors

If the server detects an error condition, it will report the error using the tag of the Command it was executing when the error was detected.

The server will abort all pending transactions.

The client must invoke the ClearAllErrors() method before the server can continue processing commands.

Further details regarding error handling are given in Section 8.

6.3 Method Syntax

The reference for this description is the C++ class definition that is part of this documentation. Please note that in the class description the first argument of all methods is Tag. This argument is converted into a CommandTag or EventTag as described before and is therefore not part of this documentation (see Server::FormatTag() method).

6.3.1 Server Methods

A session defines the time period after the client has sent a StartSession() until the client sends an EndSession() to the server.

Several states are preserved when the server is shut down, e.g. the active tool.

6.3.1.1 StartSession()

After having completed the connection between client and server on the TCP/IP level (see section 9.2) the StartSession method initiates the connection between client and server. The server can be sure that the client will invoke StartSession() only once during a session.

- StartSession()

Parameters None.

Data None.

Errors None.

Remarks The server may for example use this method to perform initial checks
Like which tool is active, ...

The method does not perform any initializations, which means that the server is in a state that was left after power up or in a state that was left over from the previous session. The client can be sure that no events or daemons are pending from the session before.

6.3.1.2 EndSession()

The client invokes this method to end a session between client and server.

The client must close the TCP/IP connection (see section 9.3) after the transaction is complete.

- EndSession()

Parameters The method has no parameters.

Data None.

Errors No errors are returned.

Remarks The method must make sure that all daemons are stopped and no events are sent after it completes. The following states of the server are preserved upon connection of the next client:

Active tool

Active coordinate system

6.3.1.3 StopDaemon(..)

The client invokes this method to stop a daemon identified by its EventTag.

- StopDaemon(EventTag)

Parameters EventTag of daemon to be stopped.

Data None.

Errors 0513: Daemon Does Not Exist.

6.3.1.4 StopAllDaemons()

The client invokes this method to stop all daemons.

- StopAllDaemons()

Parameters None.

Data None.

Errors None.

Remarks The method must make sure that all daemons are stopped and no events are sent after it completes.

6.3.1.5 AbortE()

The client invokes this method to abort all pending transactions and if possible the current one.

- AbortE()

Parameters None.

Data None.

Errors None.

Remarks The client must invoke the ClearAllErrors() method before the server will process new methods.

On receiving an AbortE command, the server must:

- (a) stop all motion as soon as possible,
- (b) stop executing any currently executing commands,
- (c) not start any pending commands (those for which an Ack has been sent but for which execution has not yet started), and
- (d) stop sending data responses for any currently executing commands.

For currently executing commands, the server must send either a TransactionComplete (for all event commands and any other commands that are completed) or an error "Transaction aborted" for non-event commands that are not complete. For pending commands, the server must send an error

"Transaction aborted".

The AbortE command itself is not to be reported complete until the responses just described have been sent. After sending an AbortE command, the client must not send any other commands until a TransactionComplete has been received in response to the AbortE. The next command sent by the client must be a ClearAllErrors command. If the server receives any other command following an AbortE, it must send an error response using error 0008 "Protocol error".

6.3.1.6 GetErrorInfo()

The client invokes this method to retrieve the error-information stored in the server.

- GetErrorInfo()

Parameters	Error-Number.
Data	Strings.
Errors	None.

6.3.1.7 ClearAllErrors()

The client invokes this method to enable the server to recover from an error.

- ClearAllErrors()

Parameters	None.
Data	None.
Errors	None.

Examples

Client to Server	Server to Client	Comment
00051 GoTo(X(1000))		
	00051 &	
00052 GoTo(Y(300))		
	00052 &	
		The client wants to abort the moves and sends
00053 AbortE()		
	00053 &	
	00051 ! Error(2,0006,GoTo,“Transaction aborted”)	
	00051 %	
	00052 ! Error(2,0006,GoTo,“Transaction aborted”)	
	00052 %	
	00053 %	
		If the client now sends
00054 Get(X())		
	00054 &	

	00054 ! Error(2,0511,Get,"Error processing method")	
	00054 ! Error(2,0514,Get,"Use clear all errors to continue")	
	00054 %	the server will still be in an error state
00055 ClearAllErrors()		
	00055 &	
	00055 %	the server is now ready to accept new method calls
00056 Get(X())		
	00056 &	
	00056 # X(23)	
	00056 %	

6.3.1.8 Information for handling properties

Each object of the system has to provide functionality to support the following functions.

- GetProp(), GetPropE()
- SetProp()
- EnumProp(), EnumAllProp()

6.3.1.9 GetProp(..)

The client uses this method to query properties of the system.

- GetProp(..)

Parameters An enumeration of one or more of the following methods can be argument.

Tool.PtMeasPar.Speed()
Tool.GoToPar.Accel()
or other methods that return properties.

Data The format is defined by the method enumerated.

Errors Errors of the enumerated methods.

6.3.1.10 GetPropE(..)

The client uses this method to query the position of the active tool. GetPropE is handled with a high priority. See GetProp().

6.3.1.11 SetProp(..)

The client uses this method to set properties of the system.

- SetProp(..)

Parameters An enumeration of one or more of the following methods can be argument.

Tool.PtMeasPar.Speed(100)
Tool.GoToPar.Accel(10)
or other methods that set properties.

Data The format is defined by the method enumerated.

Errors Errors of the enumerated methods.

6.3.1.12 EnumProp(..)

The client uses this method to query properties of the system. It returns the name of the type of a property

- EnumProp(..)

Parameters	A pointer to an object, e.g. parameter block.
	Tool.PtMeasPar() Tool.GoToPar()
Data	Returns the names of all values The client can use the type information provided to check, if the returned name is a value or a property. The property type is returned “Number” “String” “Property” ! Means class which has own properties See example 7.7.
Errors	Errors of the enumerated methods.

6.3.1.13 EnumAllProp(..)

The client uses this method to query properties of the system. It returns the names and types of the immediate children of a property.

➤ EnumAllProp(..)

Parameters	A pointer to an object, e.g. parameter block.
	Tool()
Data	Returns the names of all values and the names of all child properties of the property. The client can use the type information provided to check, if the returned name is a value or a property. The property type is returned “Number” “String” “Property” See example 7.7 for EnumProp(..).
Errors	Errors of the enumerated methods.

6.3.2 DME Methods

6.3.2.1 Home()

The client uses this method to home the machine. The server must be homed before the client can invoke methods that move the machine. The homing position is predetermined.

- Home()

Parameters None.
Data None.
Errors 1005: Error During Home.

6.3.2.2 IsHomed()

The client uses this method to query if the machine is homed.

- IsHomed()

Parameters None.
Data IsHomed(Bool).
 Bool = 0 not homed
 Bool = 1 is homed
Errors None.
Remark To get the information of one axis GetProp() functionality should be used.

6.3.2.3 EnableUser()

The client uses this method to enable user interaction with the machine.

- EnableUser()

Parameters None.
Data None.
Errors None.
Remarks The method will have arguments in the next version to allow the client to enable only a subset of the user interface elements like specific keys or joysticks only.

6.3.2.4 DisableUser()

The client uses this method to disable user interaction with the machine.

- DisableUser()

Parameters None.
Data None.
Errors None.
Remarks The server calls this method implicitly whenever the client calls a method that physically moves the machine.

6.3.2.5 IsUserEnabled()

The client uses this method to query if the user is enabled.

- IsUserEnabled()

Parameters	None.
Data	IsUserEnabled(Bool). Bool = 0 user is disabled Bool = 1 user is enabled
Errors	None.
Remarks	The client should check if the user is enabled after each StartSession() and not rely on a default.

6.3.2.6 OnPtMeasReport(..)

The client uses this method to define which information the server should send to the client when the server has completed the PtMeas command.

- OnPtMeasReport(..)

Parameters	The enumeration may not be empty. See parameters used at command Get() , section 6.3.2.11
Data	None.
Errors	0510: Bad Property .
Remarks	The server will send a report after the PtMeas command has completed.

6.3.2.7 OnMoveReportE(..)

This is a command for the Fast Queue!

The client uses this method to define which information the server should send to the client while the machine is moving.

- OnMoveReportE(..)

Parameters	Time(s), Dis(d), ... See parameters used at command Get() , section 6.3.2.11 If the enumeration is empty, no reports are sent.
Data	None.
Errors	0510: Bad Property .
Remarks	The server will send a report if the time interval s has elapsed or the machine has moved more than d millimeters. The report frequency must not be higher than 10 Hz.
Example	OnMoveReportE(Time(0.5), Dis(0.2), X(), Y(), Z())

6.3.2.8 GetMachineClass()

The client uses this method to query the type of machine.

➤ GetMachineClass()

Parameters None.

Data One of the following must be returned:
GetMachineClass("CartCMM")
GetMachineClass("CartCMMWithRotaryTable")

Errors None .

6.3.2.9 GetErrStatusE()

This is a command for the Fast Queue!

The client uses this method to query the error status of the server.

➤ GetErrStatusE()

Parameters None.

Data ErrStatus(Bool).
 Bool = 1 in error
 Bool = 0 ok

Errors None .

6.3.2.10 GetXtdErrStatus()

The client uses this method to query the extended error status of the server.

➤ GetXtdErrStatus()

Parameters None.

Data The server may send one or more lines of status information like

IsHomed(1)
IsUserEnabled(0)

...

Errors as well as one or more Errors like
 1009: Air Pressure Out Of Range
 0512: No Daemons Are Active.

6.3.2.11 Get(..)

The client uses this method to query the position of the active tool. Also temperatures and calibrated tool properties can be requested.

➤ Get(..)

Parameters An enumeration of one or more of the following methods can be argument.

X()
Y()

Z()
Tool().A()

or other methods that return axis information. Also temperatures and other dynamic properties of the system can be requested.

Data	The format is defined by the method enumerated.
Errors	Errors of the enumerated methods.
Remarks	The parameters requestable with “Get” can not be set directly.

6.3.2.12 GoTo(..)

The client uses this method to perform a multi axis move to the target position using the active tool.

➤ GoTo(..)

Parameters An enumeration of one or more of the following methods can be argument.

X(..)
Y(..)
Z(..)

Or methods that change tool properties (like Tool.A(), Tool.B()).

Data	None
Errors	Errors of the enumerated methods.
Implicit	Tool.GoToPar
Remarks	The server will move the machine, so that all axes enumerated will start to move and stop moving at the same time. The tool moves on a straight line in the MachineCsy. The movement is controlled by the Tool.GoToPar block (speed, Acceleration). Sets implicitly DisableUser().

6.3.2.13 PtMeas(..), PtMeasIJK(..)

The client uses this method to execute a single point measurement using the active tool.

Parameters necessary (Speed, clearance distance, ..) are defined by the active tool

➤ PtMeas(..)

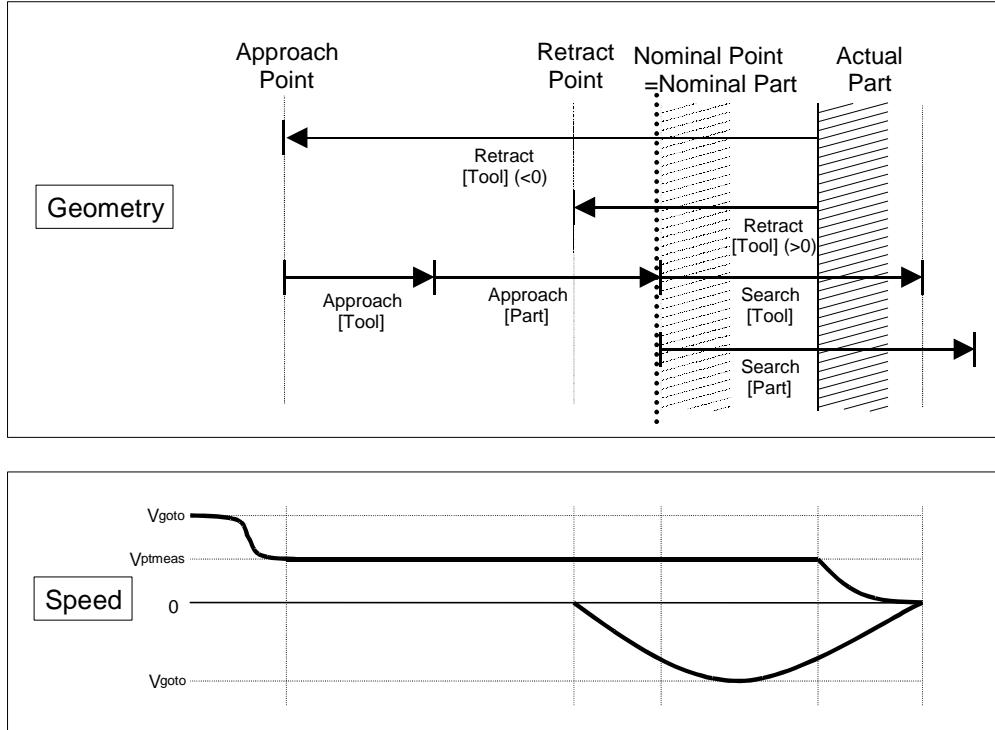
Parameters An enumeration of one or more of the following methods can be argument.

X(..)
Y(..)
Z(..)
IJK(..)

Data	As defined by OnPtMeasReport() method
Errors	1006: Surface Not Found.
Implicit	Tool.PtMeasPar
Remarks	Tool.GoToPar The PtMeas() method is processed by the server as follows:

If an IJK vector is present

- The vector I, J, K is normalized
- The nominal position X, Y, Z is shifted in the I, J, K direction by the following values:
 - Part.Clearance() used for the actual approach
 - (Tool.Approach() used as minimum for warning)
 - Tool.Radius()



This new position is called approach position.

- The server moves the machine to the approach position. This move is executed like an implicit GoTo().
- The nominal point X, Y, Z is shifted in the opposite I, J, K direction by the value of Tool.Search(). This position is called the end of search position.
- The server moves the machine to the end of search position using the PtMeasPar of the Tool().
- If the tool has part contact during this move the server latches the position of the center of the ActTool and reports to the client as defined by OnPtMeasReport().

- After contact the server will move the machine according to the value of Tool.Retract() using Tool.GoToPar for the move. If Tool.Retract() is greater or equal zero, the server will shift the contact position in the IJK direction by this value and move the machine to this position. If the Tool.Retract() is less than zero, the server will move back to the approach position as defined before.

If an IJK vector is not set by this command

- The I, J, K vector is defined as the direction from the nominal point X, Y, Z to the last position before invoking this method. F.I. the position of the last GoTo command.
- The following procedure is executed as if the I, J, K, was given by the client. See above.

Note that the end of search position may be outside the move limits, but the part surface inside. In this case the server will report success if the surface is still inside or ErrorMoveOutOfLimits. This behaviour differs from the GoTo() method.

Sets implicitly DisableUser().

6.3.2.14 Information for Tool Handling

To handle special tool behaviours the following tools are defined (predefined, reserved names)

BaseTool	! Holds the default DME capabilities, e.g. speed, acceleration
RefTool	! Supports all standard tool properties
NoTool	! Can only move but not measure
UnDefTool	!

6.3.2.15 Tool()

The client uses this method to select a pointer to the actual activated tool. It can also return a pointer to NoTool! See Example 7.6.

➤ Tool()

Parameters	None.
Data	None.
Errors	

6.3.2.16 FindTool(..)

The client uses this method to get a pointer to a tool with a known name. See Example 7.7.

➤ FindTool("Too1")

Parameters	Name of tool to search for.
------------	-----------------------------

Data	FoundTool.
Errors	1502: Tool Not Found.

6.3.2.17 FoundTool()

This method acts as a pointer to a tool with a known name selected by FindTool("xxx"). FoundTool() is only valid after a call to FindTool(). See Example 7.7.

- FoundTool()

Parameters	None.
Data	None.
Errors	
Remark	Pointer can also be NoTool!

6.3.2.18 ChangeTool(..)

The client uses this method to change the tool by ProbeChanger or manually.

- ChangeTool("Tool2")

Parameters	Name of the tool to activate.
Data	None.
Errors	1502: Tool Not Found.
Remark	If an error occurs during the execution of ChangeTool(), the client is responsible to ask the server which tool is active then.

6.3.2.19 SetTool(..)

The client uses this method to force the server to assume a given tool is the active tool.

- SetTool("Tool2")

Parameters	Name of the tool to set.
Data	None.
Errors	1502: Tool Not Found.
Remark	If an error occurs during the execution of SetTool(), the client is responsible to ask the server which tool is active then.

The server assumes the active tool is "Tool2"

6.3.2.20 AlignTool(..)

The client uses this method to force the tool to orientate according to the given vector(s).

- AlignTool(i1,j1,k1, alpha)
- AlignTool(i1,j1,k1,i2,j2,k2, alpha, beta)

Parameters One normalized vector (i1, j1, k1). This vector is anti parallel to the main axis of the tool (away from the surface).

Two normalized vectors (i1, j1, k1, i2, j2, k2). The first vector is anti parallel to the main axis of the tool (away from the surface). The second vector describes the orientation in the working plane.

Maximal allowed error angles (alpha, beta) in which the found orientation may differ from the desired one. In case the angle differs, ToolNotFound is returned. In case alpha or beta are zero no error check is performed.

Data Returns vectors (same number as set) which describe the reached alignment.
Errors 1502: Tool Not Found.

6.3.2.21 GoToPar()

This method acts as a pointer to the GoToParameter block of the DME.

➤ GoToPar()

Parameters None.

Data pointer.

Errors

Remark

6.3.2.22 PtMeasPar()

This method acts as a pointer to the PtMeasParameter block of the DME.

➤ PtMeasPar()

Parameters None.

Data pointer.

Errors

Remark

6.3.2.23 EnumTools()

The client uses this method to query the names of the available tools. It returns a list of names.

➤ EnumTools()

Parameters None

Data Returns the names of all values. F.I.:
00014 &
00014 "ReferenceProbe"
00014 "NormalTool"
00014 "Conf1.Tip1"
00014 "Conf1.Tip2"

...
00014 "SpecialTool"
00014 %
Errors Errors of the enumerated methods.

6.3.3 CartCMM Methods

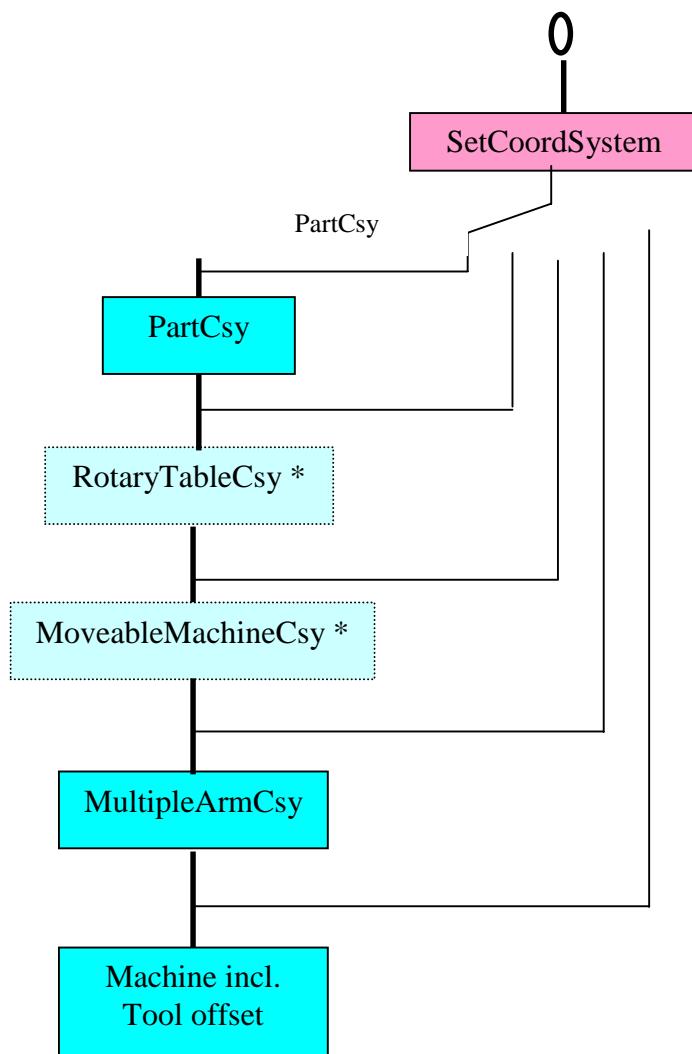
Each CartCMM implements a cartesian machine coordinate system.
Based on this coordinate system the following depend on it:

MachineCsy
MoveableMachineCsy
MultipleArmCsy
PartCsy

The multiple arm transformation is implemented also on bridge type or single arm machines

* The RotaryTableCsy (handling rotary table) and the MoveableMachineCsy (handling movable measurement equipment, mechanical or optical) are listed here because of consistency reasons of the transformation chain.

Picture 21: Transformation chain model



As example the transformation of a point in machine coordinates (x, y, z) to a point in multiple

arm coordinates (x',y',z') is calculated as follows.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \begin{pmatrix} x - x_0 \\ y - y_0 \\ z - z_0 \end{pmatrix}$$

In this example x0, y0, z0 and the coefficients m11 ... m33 are calculated as follows from the arguments of the SetCsyTransform(MultipleArmCsy, X0, Y0, Z0, Theta, Psi, Phi)

To create the Euler Angles Theta, Phi, Psi and vice versa the rotation matrix see Appendix A.4.2.

6.3.3.1 SetCoordSystem(..)

The client uses this method to select the coordinate system it wants to work with.

➤ SetCoordSystem(..)

Parameters	One of the following: MachineCsy MoveableMachineCsy MultipleArmCsy PartCsy.
Data	None.
Errors	0509: Bad Parameter.
Remarks	The parameters are considered to be enums and must not be enclosed in double quotes.

6.3.3.2 GetCoordSystem()

The client uses this method to query the server which coordinate system is selected..

➤ GetCoordSystem()

Parameters	None.
Data	CoordSystem(Arg).
	Arg can be one of the following: MachineCsy MoveableMachineCsy MultipleArmCsy PartCsy.
Errors	None.

6.3.3.3 GetCsyTransformation(..)

The client uses this method to get the enumerated coordinate transformation back from the server.

➤ GetCsyTransformation(Enumerator)

Parameters	Enumerator: PartCsy
------------	---------------------

JogDisplayCsy
 JogMoveCsy
 SensorCsy
 MoveableMachineCsy
 MultipleArmCsy.

Data	GetCsyTransformation(X0, Y0, Z0, Theta, Psi, Phi).
Errors	None.
Remarks	The definition of the relation between transformation matrix and parameters is given in the C++ class definition. See Section 10.4.2.

6.3.3.4 SetCsyTransformation(..)

The client uses this method to replace the enumerated coordinate transformation.

➤ SetCsyTransformation(Enumerator, X0,Y0,Z0, Theta, Psi, Phi)

Parameters	Enumerator: PartCsy JogDisplayCsy JogMoveCsy SensorCsy MoveableMachineCsy MultipleArmCsy.
------------	--

X0, Y0, Z0 define the zero point of the machine coordinate system in part coordinates. Theta, Psi and Phi are Euler angles that define the rotation matrix of the transformation.

Data	None.
Errors	1007: Theta Out Of Range.
Remarks	See Section 10.4.2. Theta must be in the range of 0..180 degrees. Psi and Phi should be normalized (modulo 360) by the server.

6.3.3.5 X()

➤ X()

Parameters	None.
Data	X(x).
Errors	1503: Tool Not Defined 0509: Bad Parameter.
Remarks	This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.6 Y()

➤ Y()

Parameters	None.
Data	Y(y).
Errors	1503: Tool Not Defined 0509: Bad Parameter.

Remarks This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.7 Z()

➤ Z()

Parameters None.
Data Z(z).
Errors 1503: Tool Not Defined
 0509: Bad Parameter.
Remarks This method can only be invoked as an argument of a Get or OnReport method.

6.3.3.8 IJK()

➤ IJK()

Parameters None.
Data IJK(i,j,k).
Errors 0508: Bad Context.

Remarks i,j,k define a direction vector in the actual coordinate system. The vector is not necessarily normalized. Its values are tool dependent. If the client normalizes the vector it should point out of the part material.
This method can only be invoked as an argument of OnPtMeasReport().

6.3.3.9 X(..)

➤ X(x)

Parameters target x position.
Data None. ...
Errors 2500: Machine Limit Encountered
 2504: Collision
 0508: Bad Context.
Implicit Tool.GoToPar
Remarks This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.10 Y(..)

➤ Y(y)

Parameters target y position.
Data None. ...
Errors 2500: Machine Limit Encountered
 2504: Collision
 0508: Bad Context.
Implicit Tool.GoToPar
Remarks This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.11 Z(..)

➤ Z(z)

Parameters target z position.
Data None. ...
Errors 2500: Machine Limit Encountered

	2504: Collision
	0508: Bad Context.
Implicit	Tool.GoToPar
Remarks	This method can only be invoked as an argument of a GoTo or PtMeas method. If the server detects a MoveOutOfLimits condition, the machine will not move.

6.3.3.12 IJK(..)

➤ IJK(i,j,k)

Parameters	i,j,k define the X,Y,Z values of a vector.
Data	None.
Errors	0508: Bad Context. 1010: Vector Has No Norm.
Remarks	i,j,k define a direction vector in the machine coordinate system. The vector is not necessarily normalized. Before using the vector, the server must normalize it. This method can only be invoked as an argument of another method.

6.3.3.13 R()

The client uses this method to query the position of the rotary table. Implementation in CartCMMWithRotTbl.

➤ R()

Parameters	None.
Data	R().
Errors	0509: Bad Parameter.
Remarks	This method can only be invoked as an argument of a Get or OnReport method. The setting of the rotary table must be done by AlignPart!

6.3.4 ToolChanger Methods

Each CMM implements one instance of the ToolChanger class to install and change tools. The methods are available, and described here in the DME section, because there is exactly one instance.

6.3.5 Tool Methods (Instance of class KTool)

Each CMM implements a class KTool to contain the properties of the tool and the methods to handle them.

6.3.5.1 GoToPar()

This method acts as a pointer to the GoToParameter block of this instance of KTool.

➤ GoToPar()

Parameters None.
Data pointer.
Errors
Remark

6.3.5.2 PtMeasPar()

This method acts as a pointer to the PtMeasParameter block of this instance of KTool.

➤ PtMeasPar()

Parameters None.
Data pointer.
Errors
Remark

6.3.5.3 ReQualify()

The client uses this method to requalify ActTool.

➤ ReQualify()

Parameters None, ActTool is used.
Data None.
Errors Error messages during calibration.
Remark

6.3.6 GoToPar Block

Each parameter block contains information for

Speed and
Accel.

Each of these physical values are split in

Min
Max
Act and
Def.

There is the parameter-block of the active tool accessible via the DME and a parameter-block associated with each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h. The methods to access the values are described in the examples 7.7

The application can not set the actual values outside the range defined by Min. and Max values. The application can not change Min and Max. If trying to set the actual value outside the defined range, the Max or Min value is used.

6.3.7 PtMeasPar Block

Each parameter block contains information for

Speed
Accel
Approach
Search
Retract.

Each of these physical values are split in

Min
Max
Act and
Def.

There is the parameter-block of the active tool accessible via the DME and a parameter-block associated to each tool. The access to the parameter-blocks is described in the object model 5.9 and in the header file of toolchanger.h. The methods to access the values are described in the examples 7.7.

The application can not set the actual values outside the range defined by Min. and Max values. The application can not change Min and Max. If trying to set the actual value outside the defined range, the Max or Min value is used.

The ABCGoToPar and ABCPtMeasPar are mentioned in the full object model. This is because of symmetry reasons. Because actual rotational heads cannot be controlled in that manner it is not necessary to implement this actually.

6.3.8 A(), B(), C()

The client uses this method to query one or more rotational axis of the ActTool. The reference to the rotational axis can be used single. Implementation in ToolAB or ToolABC.

➤ A()

Parameters	None.
Data	A().
Errors	1503: Tool Not Defined 0509: Bad Parameter.
Remarks	This method can only be invoked as an argument of a Get or OnReport method. The usage from the interface is Tool.A()... See examples chapter 7.

6.3.9 A(..), B(..), C(..)

For internal and symmetry reasons. The setting of the rotational axis should be done by AlignTool. Only this handling guarantees compatibility between the implementations!

➤ A(a)

7 Additional Dialog Examples

7.1 StartSession

Client to Server	Server to Client	Comment
		Server and Client must be booted up previously
00001 StartSession		Client connects to server
	00001 &	Server sends acknowledge
	00001 %	Server sends transaction complete

7.2 Move 1 axis

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy, 10, 20, 30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to and from part coordinate system
	00010 &	
	00010 %	
00011 GoTo(X(100))		Move now in part coordinate system
	00011 &	
	00011 %	

7.3 Probe 1 axis

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),To ol.A())		Client defines format for probing result. Valid for every PtMeas command from now on.
	00014 &	
	00014 %	
00015 PtMeas(X(200))		Uses standard method in CartCMM
	00015 &	
	00015 # X(199.998),Y(250.123),Z (300.002),Tool.A(45)	Probing result from server
	00015 %	

7.4 Move more axes in workpiece coordinate system

Client to Server	Server to Client	Comment
00009 SetCsyTransformation(PartCsy,10, 20,30, 0, 0, 0)		Set transformation for part coordinate system
	00009 &	
	00009 %	
00010 SetCoordSystem(PartCsy)		Select transformation to part coordinate system
	00010 &	
	00010 %	
00011 Goto(X(100),Y(150),Z(200)) 00011 Goto(X(100),Y(150),Z(200),R(180))		Move with more axes Alternatively
	00011 &	
	00011 %	

7.5 Probe with more axis

Client to Server	Server to Client	Comment
00014 OnPtMeasReport(X(),Y(),Z(),To ol.A())		Valid for every PtMeas command
	00014 &	
	00014 %	
00015 PtMeas(X(200),Y(250),Z(300)) 00015 PtMeas(X(200),Y(250),Z(300),I JK(0,0,1)) 00015 PtMeas(X(200),Y(250),Z(300),I JK(0,0,1),R(180))		Uses standard method in CartCMM Alternatively, with approaching vector Alternatively, with approaching vector and rotary table
	00015 &	
	00015 # X(199.998),Y(250.123),Z (300.002),Tool.A(45)	Result
	00015 %	

7.6 Set property

Client to Server	Server to Client	Comment
00015 SetProp(Tool.PtMeasPar.Speed(100))		Set probing speed of active tool
	00015 &	
	00015 %	

7.7 Get, read property

All properties that are represented as strings are exchanged using double-quotes, e.g. “This is my probe”

Client to Server	Server to Client	Comment
00014 EnumProp(Tool.PtMeasPar())		Get ActTools PtMeas Property list
	00014 &	
	00014 # “Speed”, “Number”	
	00014 # “Accel”, “Number”	
	
	00014 # “Approach”, “Number”	
	00014 %	
00015 GetProp(Tool.PtMeasPar.Speed())		Request for getting probing speed of active tool
	00015 &	
	00015 # Tool.PtMeasPar.Speed(10 0)	
	00015 %	
00016 FindTool(“Probe1”)		Search pointer to Probe 1
	00016 &	
	00016 %	
00017 GetProp(FoundTool.PtMeasPar. Speed())		Get Probing speed of Probe1
	00017 &	
	00017 # FoundTool.PtMeasPar.Sp eed(100)	
	00017 %	

8 Error Handling

- Each transaction can generate multiple error messages.
- These messages are headed by the same tag number.

8.1 Classification of Errors

Please note that all error numbers are UnsInt (unsigned integers).

F1 :== UnsInt
F2 :== UnsInt
F3 :== String
Text :== String

ErrorResponse :== Tag | ETag “ ! Error“ {“ “}“(“ {“ “}F1 {“ “} “,” {“ ”} F2 {“ “} “,” {“ ”} F3{“ “} “,” {“ ”} Text “)“

0003 ! Error(F1, F2, F3, Text)

F1: Error severity classification

0: Info

1: Warning

2: Error, client should be able to repair the error

3: Error, user interaction necessary

9: Fatal server error

Errors with classification higher or equal 2 require ClearAllErrors().

F2: Error numbers, 0000-4999, defined by I++ DME

 5000-8999 definable from server

 9000-9999 definable from client

F3: I++ recommends to serve here the name of the error causing method. This means the name of the function in the server implementation that reported the error.

8.2 List of I++ predefined errors

Classification in Field F2

0000-0499 Protocol, syntax error

0500-0999 Error generated during execution in DME (see object model)

1000-1499 Error generated during execution in CartCMM... (see object model)

1500-1999 Error generated during execution in ToolChanger (see object model)

2000-2499 Error generated during execution in Tool... (see object model)

2500-2999 Error generated during execution in Axis (see object model)

Defined errors:

Severity class	Error No.	Text
0	0000	Buffer full

2	0001	Illegal tag
2	0002	No space at pos. 6
2	0003	Reserved
2	0004	Reserved
1	0005	Suspend communication
2	0006	Transaction aborted (Use ClearAllErrors To Continue)
3	0500	Emergency stop
3	0501	Unsupported command
3	0502	Incorrect parameters
9	0503	Controller communications failure
1	0504	Parameter out of range
3	0505	Parameter not recognized
3	0506	Parameter not supported
3	0507	Illegal command
3	0508	Bad context
3	0509	Bad parameter
3	0510	Bad property
2	0511	Error processing method
1	0512	No daemons are active
2	0513	Daemon does not exist
2	0514	Use ClearAllErrors to continue
3	1000	Machine in error state
2	1001	Illegal touch
9	1002	Axis does not exist
2	1003	No touch
9	1004	Number of angles not supported on current device
2	1005	Error during home
2	1006	Surface not found
3	1007	Theta out of range
3	1008	Target position out of machine volume
3	1009	Air pressure out of range
2	1010	Vector has no norm
3	1500	Failed to re-seat head
3	1501	Probe not armed
3	1502	Tool not found
3	1503	Tool not defined
3	2000	Tool not calibrated
2	2001	Head error excessive force
3	2002	Type of probe does not allow this operation
3	2500	Machine limit encountered [Move Out Of Limits]
3	2501	Axis not active
2	2502	Axis position error
9	2503	Scale read head failure
3	2504	Collision
2	2505	Specified angle out of range
2	2506	Part not aligned

9 Miscellaneous Information

9.1 Coordination of company related extensions

The goal of each company should be to bring extensions to the standard. This is highly recommended by the I++ DME Team. To allow fast uncoordinated developments tryouts before that, specific name spaces are reserved.

Company specific extensions can be applied to public methods and properties of the server.

The following mechanism is offered:

Commands in Class DME beginning with

BS. Are Brown&Sharpe proprietary

CZ. Are Carl Zeiss proprietary

WP. Are Wenzel Präzision proprietary

MI. Are Mitutoyo proprietary...

XX. Are company short terms

The handling of properties is done in the same way

SetProp(XX....)

GetProp(XX....) is XX company proprietary.

Additional short terms can be requested from the I++ DME team.

9.2 Initialization of TCP/IP protocol-stack

After CMM power up the server will create the application port in listen mode.

When the client is started, it will send a connection request to the application port created by the server. The server will confirm the connection and is now ready to work with the client.

9.3 Closing TCP/IP connection

When the client no longer needs the server it will close the connection.

The driver will then listen on the application port for new incoming connection requests.

9.4 EndSession and StartSession

After re-starting a session all previous defined properties are valid again.

9.5 Pre-defined Server events

The following server events are predefined. Please note that all these events are transmitted with tag number E0000 to the client.

9.5.1 KeyPress

Data KeyPress("NameOfKey")

Remarks The server sends this event, if the user is enabled and when the user has pushed button on the jog box.

9.5.2 Clearance or intermediate point set

Data GoTo(...)

Remarks The server sends this event, if the user is enabled and when the user has pushed the “Clearance Point” button on the jog box.
The GoTo format is defined by OnMoveReport().

9.5.3 Pick manual point

Data PtMeas(...)

Remarks The server sends this event, if the user is enabled and when the user manually picked a point.
The Ptmeas format is defined by OnPtMeasReport().

9.5.4 Change Tool request

Data ChangeTool(“ToolName”)

Remarks The server sends this event, if the user is enabled and when the user has pushed a button on the jogbox that selects a new tool.
This event is only a request. The client has to decide if it should send a ChangeTool() command to execute the change.
Note that this command does not change the tool.

9.5.5 Set property request

Data SetProp()

Remarks The server sends this event, if the user is enabled and when the user has pushed a button on the jogbox that changes a property.
This event is only a request. The client has to decide if it should send a SetProp() command to execute the change.
Note that this command does not change the property.

9.5.6 Additional defined keys

The following NameOfKeys are additional defined:

“Done” // signals an operation should be finished
“Del” // delete a function call or a measured point...
“F1” ... “Fn” // key code of softkeys

9.6 Reading part temperature

In the appended c++ header files, the temperature is a property of the class part. This is actual not public, but will become in further revisions. The access will be by

GetProp(Part.Temperature)

9.7 Links to important sites

Link to IA.CMM where this spec, the Rose model files and also the C++ headerfiles can be downloaded:

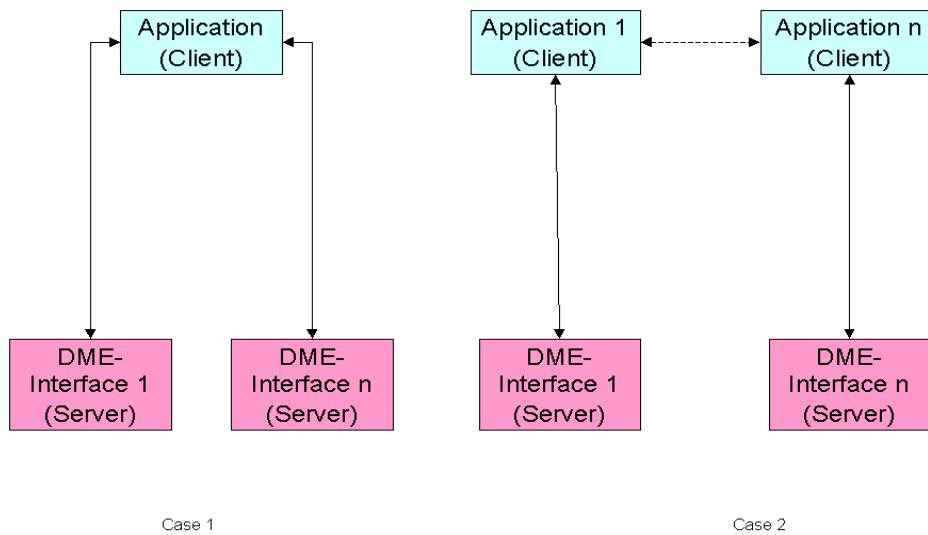
http://www.iacmm.org/stand_main.htm

Link to NIST where the DME testbed can be downloaded:

http://www.isd.mel.nist.gov/projects/metrology_interoperability/resources.html

10 Multiple arm support

Picture 22: Multiple arm equipment



- For each column a single I++ DME interface is required.
- The disposition of a feature to be measured on a specific column, the synchronisation of the columns (including collision detection between the columns) and the combination of the results is part of the application task.
- The vendor of the multiple arm system has to provide an application to build the coupling transformation. This can be a stand alone application or an integrated part of the DME interface.
- The following commands are used to set and get these transformations
`SetCsyTransformation(MultipleArmCsy,.....)`, `GetCsyTransformation(MultipleArmCsy)`

A coupling tool is used to define the multiple arm coordinate system for each column. The coupling tool may be a special tool or the application itself.

The coupling tool measures an artefact to calculate the `MultipleArmCsy`. The sequence of the measurement is as follows.

- The coupling tool measures the artifact using column 1.
- The coupling tool measures the artifact using column 2.
- The coupling tool calculates the 2 transformations used for column 1 and 2.
- The coupling tool sends the transformation for column 1 using a
`SetCsyTransformation(MultipleArmCsy,.....)` command to DME of column 1.
- The coupling tool sends the transformation for column 2 using a
`SetCsyTransformation(MultipleArmCsy,.....)` command to DME of column 2.

11 Scanning

11.1 Preliminaries

11.1.1 Hints:

- Hints are used to communicate properties of the part to the DME
- The only use for Hints is to optimize the execution of a measuring process.
- Hints are not mandatory; the DME must be able to execute without the interpretation of a given hint.

The definition of the scanning commands is independant of the type of sensor, F.E. tactile, measuring. Tactile sensors may emulate the functionality of measuring sensors. The algorithm is not part of the spec.

11.1.2 OnScanReport

Defines properties reported while scanning.

➤ **OnScanReport()**

Parameters	Enumeration of properties reported for a scan
Data	
Errors	Bad property
Remarks	<p>Besides properties like X(), Y(), Z() the scan can report a Q() property that defines a “quality” for a scan point returned to the client by the DME.</p> <p>The Q() property is a numeric value between 0 and 100 indicating the “quality” of the measured point. A value of 0 defines a “good” point. Depending on the tool used to scan, values from 1 to 100 indicate a lower quality and reliability of the points. A value of 100 mark bad points.</p> <p>If points are out of the tools measuring range, the DME may decide to flag them as “bad points” or stop the scan with an error.</p> <p>To increase system performance the transmit of the already measured data from the DME to the client is sensful while the execution of the scanning command is still in process.</p> <p>To prevent overhead (TCP/IP and other...) in returning each measuring value as an own string, the scanning results can be blocked. Multiple measuring results can be returned in one string. The number of values in a return string must be multiple of the definition done by OnScanReport. See example in 11.4.</p>

11.2 Scanning known contour

11.2.1 ScanOnCircleHint

The ScanOnCircleHint command defines expected deviations of the measured circle from the nominal circle. The displacement and the form can be used by the DME to optimize the execution of the ScanOnCircle command.

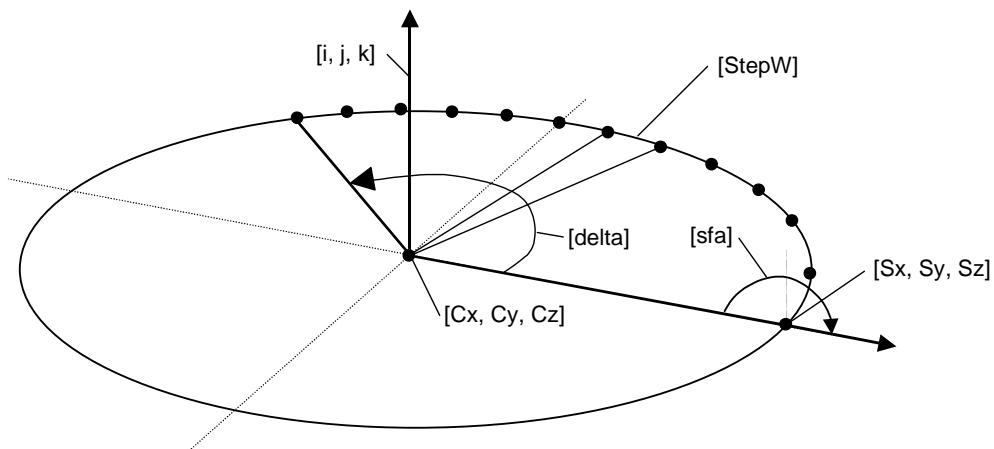
- ScanOnCircleHint (Displacement, Form)

Parameters	Displacement defines the maximum expected distance between the nominal circle center and the actual circle center. Form defines the maximum expected form deviation calculated by Gauss of the circle.
------------	---

11.2.2 ScanOnCircle

- ScanOnCircle(Cx, Cy, Cz, Sx, Sy, Sz, i, j, k, delta, sfa, StepW)

Parameters	Cx, Cy, Cz	is the nominal center point of the circle
	Sx, Sy, Sz	is a point on the circle radius where the scan starts
	i,j,k	is the normal vector of the circle plane
	delta	is the angle to scan
	sfa	is the surface angle of the circle.
	StepW	average angular distance between 2 measured points in degrees.



Data Errors	As defined by OnScanReport
Remark	The distance between the center point (Cx,Cy,Cz) and the start point (Sx,Sy,Sz) may not be zero. The distance is the nominal radius of the circle to scan.
	The plane vector (i,j,k) must be orthogonal to the vector from the center point to the start point (start direction).

The angle delta may be positive or negative and defines the arc to scan.

Assume (i,j,k) to be the z-direction of a coordinate system and the start direction the x-direction. The reference for delta is the x-direction and the angle delta is defined in the xy plane.

The surface angle is the angle between the x-direction and the material direction in the xz plane. The surface angle is 0 for an outside circle and 180 for an inside circle. Using a surface angle enables to execute a circulat path scan on cylinders (sfa=0 or sfa=180), on planes (sfa=90 or sfa=270) and cones. In the context of this command cylinders and planes are specialized cones.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and vector derived from the surface normal in the start point.

The actual scan radius is calculated from the circle center point (Cx, Cy, Cz) and the result point of the PtMeas command. The DME will scan on a circle defined by (Cx,Cy,Cz) and the actual scan radius.

The DME will scan the arc defined by delta.

During the scan the probe will move in the cone shell defined by the PtMeas result point and the probing direction rotated around an axis defined by (Cx,Cy,Cz,i,j,k).

The DME will return approximately delta/StepW points to the client using the format defined by OnScanReport.

11.2.3 ScanOnLineHint

The ScanOnLineHint command defines expected deviations of the measured line from the nominal line. The displacement and the form can be used by the DME to optimize the execution of the ScanOnCircle command.

- ScanOnLineHint (Angle, Form)

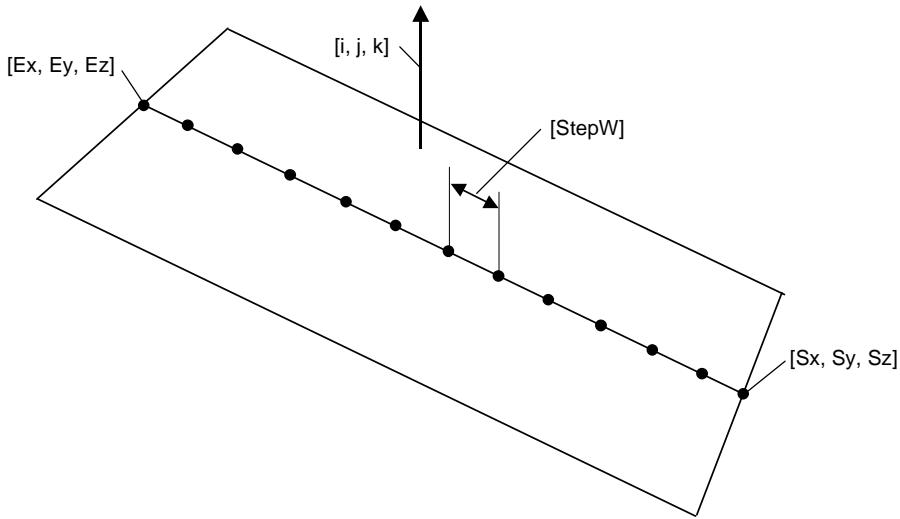
Parameters Angle defines the maximum expected angle between the nominal line and the actual line.
 Form defines the maximum expected deviation form of the Gaus calculated line.

11.2.4 ScanOnLine

- ScanOnLine(Sx,Sy,Sz,Ex,Ey,Ez,i,j,k,StepW)

Parameters Sx, Sy, Sz defines the line start point
 Ex, Ey, Ez defines the line end point

i,j,k
 StepW is the surface normal vector on the line
 average distance between 2 measured points in mm.



Data	As defined by OnScanReport
Errors	
Remark	The distance between the start point (S_x, S_y, S_z) and the end point (E_x, E_y, E_z) may not be zero. This is the distance to scan.
	The surface vector (i, j, k) must be orthogonal to the vector from the start point to the end point.

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (S_x, S_y, S_z) as point and (i, j, k) as surface normal.

The actual start point for the scan is the result point of the PtMeas command.

The DME will scan a the distance between start and end point..The scan terminates if the distance between a measured point and the actual start point is greater the the distance between (S_x, S_y, S_z) and (E_x, E_y, E_z),

During the scan the probe will move in a plane defined by (S_x, S_y, S_z) and the vector of the cross product between (i, j, k) and the direction from start to stop point.

The DME will return approximately (distance start/stop)/StepW points to the client using the format defined by OnScanReport.

11.3 Scan unknown contour

11.3.1 ScanUnknownHint

The ScanUnKnownHint command defines expected minimum radius of curvature in the unknown contour.

- ScanUnknownHint (MinRadiusOfCurvature)

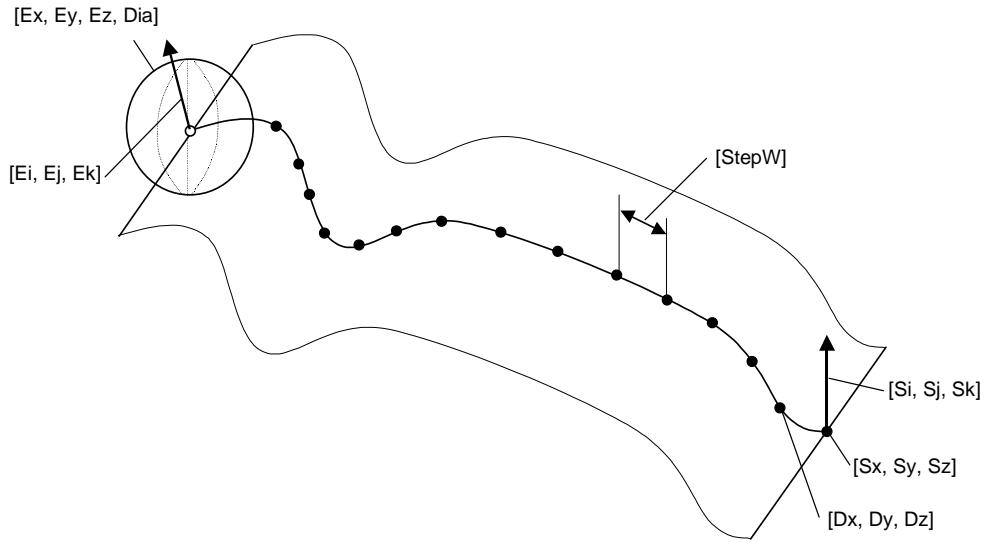
Parameters Prognosted minumum radius in the curve to measure.

11.3.2 ScanInPlaneEndIsSphere

The ScanInPlaneEndIsSphere allows to scan an unknown contur. The scan will stop if the sphere stop criterium is matched.

- ScanInPlaneEndIsSphere(Sx,Sy,Sz,Si,Sj,Sk,Dx,Dy,Dz,StepW, Ex,Ey,Ez,Dia,Ei,Ej,Ek)

Parameters	Sx, Sy, Sz	defines the scan start point
	Si, Sj, Sk	defines the surface direction in the start point
	Dx, Dy, Dz	defines the scan direction point
	StepW	is the average distance between 2 measured points
	Ex, Ey, Ez,	defines the scan end point
	Dia	define a sphere arount the end point where the scan stops.
	Ei, Ej, Ek,	defines the surface direction at the end point



Data	As defined by OnScanReport
Errors	
Remark	<p>The distance between the start point (S_x, S_y, S_z) and the direction point (D_x, D_y, D_z) may not be zero.</p> <p>The end point must be in the scanning plane defined by the start point and the normal vector defined by the cross product of (S_i, S_j, S_k) and the direction vector from start to direction point..</p>

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (S_x, S_y, S_z) as point and (S_i, S_j, S_k) as surface normal.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move within the scanning plane

The DME will stop scanning when within the stop sphere the distance between a scanned point and the sphere center has a local minimum.

If the start point is within the stop sphere, the DME will first leave the sphere and then start to checking the stop criterium.

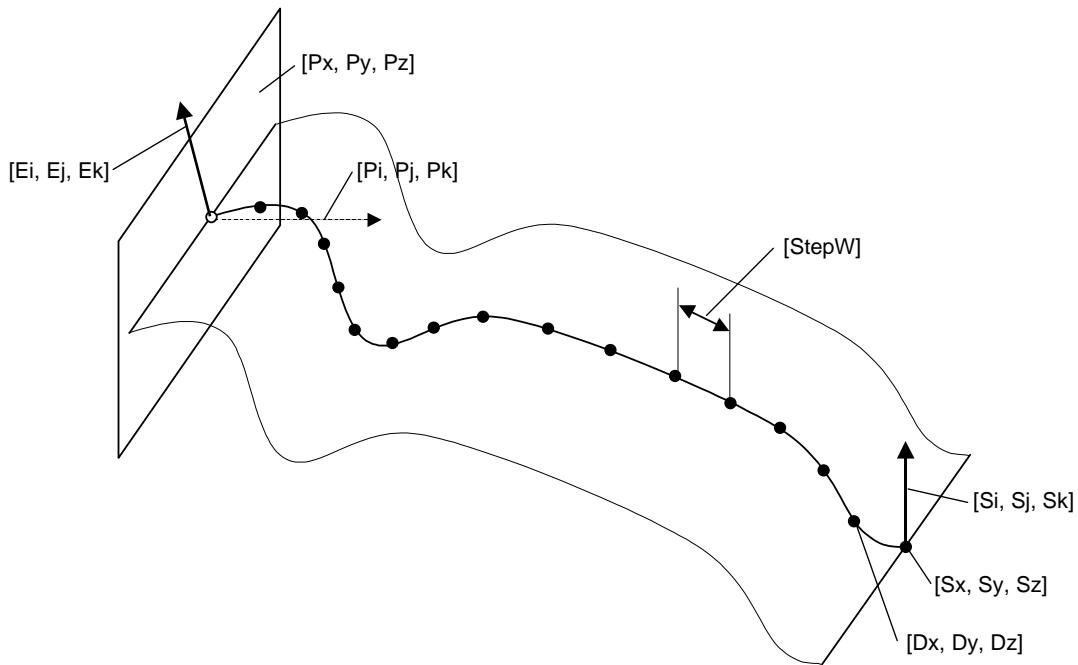
11.3.3 ScanInPlaneEndIsPlane

The ScanInPlaneEndIsPlane allows to scan an unknown contur. The scan will stop if the plane stop criterium is matched.

➤ ScanInPlaneEndIsPlane($S_x, S_y, S_z, S_i, S_j, S_k, D_x, D_y, D_z, StepW,$
 $P_x, P_y, P_z, P_i, P_j, P_k, n, E_i, E_j, E_k$)

Parameters	S_x, S_y, S_z	defines the scan start point
	S_i, S_j, S_k	defines the surface direction in the start point

Dx, Dy, Dz	defines the scan direction point
$StepW$	is the average distance between 2 measured points
$Px, Py, Pz,$	
Pi, Pj, Pk	Define a plane where the scan stops
n	Number of through the plane
Ei, Ej, Ek	surface vector at the end point



Data	As defined by OnScanReport
Errors	
Remark	<p>The distance between the start point (Sx, Sy, Sz) and the direction point (Dx, Dy, Dz) may not be zero.</p> <p>The stop plane must be perpendicular to the scanning plane defined by the start point and the normal vector defined by the cross product of (Si, Sj, Sk) and the direction vector from start to direction point..</p>

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx, Sy, Sz) as point and (Si, Sj, Sk) as surface normal.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move within the scanning plane

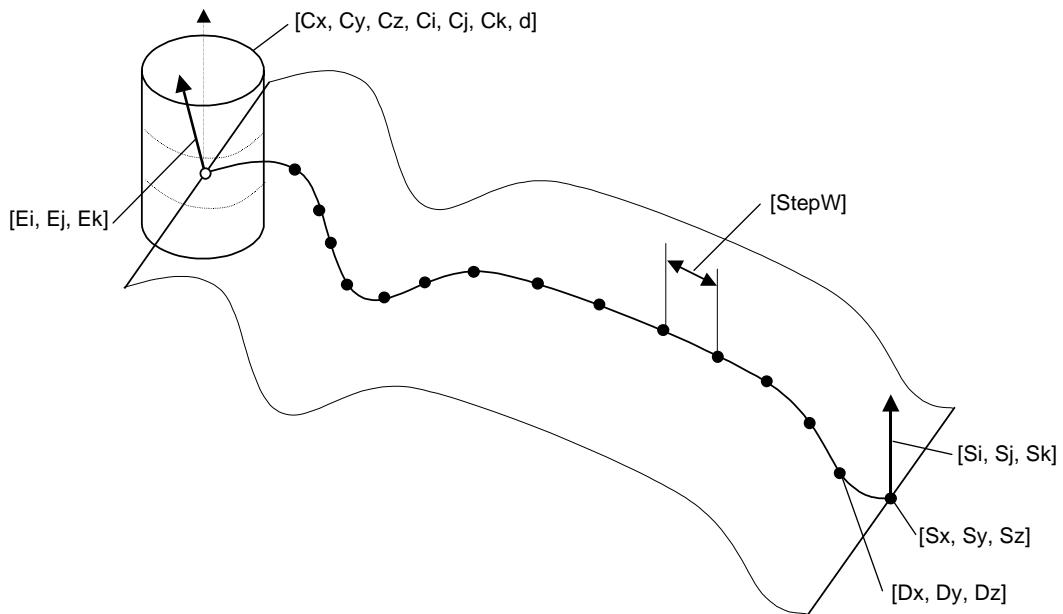
The DME will stop scanning when it passes n times through the stop plane.
The DME will start to check the stop criteria when it moved a distance that is larger than the distance between start and direction point.

11.3.4 ScanInPlaneEndIsCyl

The ScanInPlaneEndIsPlane allows to scan an unknown contur. The scan will stop if the cylinder stop criterium is matched.

- ScanInPlaneEndIsCyl(Sx,Sy,Sz,Si,Sj,Sk,dx,dy,dz,StepW,
Cx,Cy,Cz,Ci,Cj,Ck,d,n,Ei,Ej,Ek)

Parameters	Sx, Sy, Sz	defines the scan start point
	Si, Sj, Sk	defines the surface direction in the start point
	Dx, Dy, Dz	defines the scan direction point
	StepW	is the average distance between 2 measured points
	Cx, Cy, Cz, Ci, Cj, Ck, d	define a cylinder where the scan stops.
	n	Number of through the cylinder
	Ei, Ej, Ek	defines the surface vector at the end point



Data Errors	As defined by OnScanReport
Remark	<p>The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.</p> <p>The (Ci,Cj,Ck) must be parallel to the scanning plane defined by the start point and the normal vector defined by the cross product of (Si,Sj,Sk) and the direction vector from start to direction point..</p>

The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal.

The DME will start to scan into the direction from start to direction point.
 During the scan the tool center will move within the scanning plane

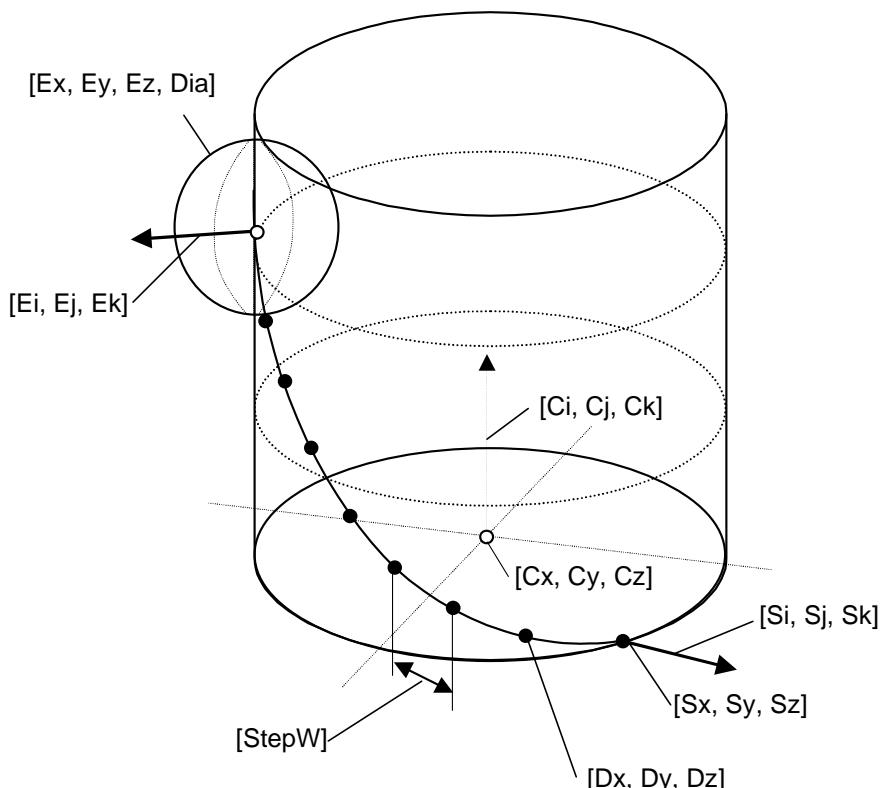
The DME will stop scanning when it passes n times through the stop cylinder.

11.3.5 ScanInCylEndIsSphere

The ScanInCylEndIsSphere allows to scan an unknown contur. The scan will stop if the sphere stop criterium is matched.

- ScanInCylEndIsSphere($C_x, C_y, C_z, C_i, C_j, C_k,$
 $S_x, S_y, S_z, S_i, S_j, S_k,$
 $D_x, D_y, D_z, StepW,$
 $E_x, E_y, E_z, Dia, E_i, E_j, E_k)$

Parameters	C_x, C_y, C_z	
	C_i, C_j, C_k	defines the axis of the cylinder
	S_x, S_y, S_z	defines the scan start point
	S_i, S_j, S_k	defines the surface direction in the start point
	D_x, D_y, D_z	defines the scan direction point
	StepW	is the average distance between 2 measured points
	E_x, E_y, E_z, Dia	Define a sphere where the scan stops
	E_i, E_j, E_k	defines the surface at the end point



Data As defined by OnScanReport

Errors	
Remark	<p>The distance between the start point projected to the cylinder axis and the start point (Sx,Sy,Sz) may not be zero and defines the diameter of the cylinder.</p> <p>During the scan the tool center will move within the surface (ScanningCylinder), that is created by rotating a line (Sx,Sy,Sz, Ci,Cj,Ck) around the cylinder axis.</p>
	<p>The distance between the start point (Sx,Sy,Sz) and the direction point (Dx,Dy,Dz) may not be zero.</p>
	<p>The scan is executed as follows:</p>

The DME will implicitly execute a PtMeas command using (Sx,Sy,Sz) as point and (Si,Sj,Sk) as surface normal.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move ScanningCylinder.

The DME will stop scanning when within the stop sphere the distance between a scanned point and the sphere center has a local minimum.
If the start point is within the stop sphere, the DME will first leave the sphere and then start checking the stop criterium.

11.3.6 ScanInCylEndIsPlane

The ScanInCylEndIsPlane allows to scan an unknown contur. The scan will stop if the plane stop criterium is matched.

➤ ScanInCylEndIsPlane(Cx,Cy,Cz,Ci,Cj,Ck,
 Sx,Sy,Sz,Si,Sj,Sk,
 Dx,Dy,Dz,StepW,
 Px,Py,Pz,Pi,Pj,Pk,n
 Ei,Ej,Ek)

Parameters	Cx, Cy, Cz	
	Ci,Cj,Ck	defines the the axis of the cylinder
	Sx, Sy, Sz	defines the scan start point
	Si, Sj, Sk	defines the surface direction in the start point
	Dx, Dy, Dz	defines the scan direction point
	StepW	is the average distance between 2 measured points
	Px, Py, Pz,	
	Pi, Pj, Pk	defnes the stop plane
	n	number of through stop plane
	Ei, Ej, Ek	surface direction at end point

Data Errors	As defined by OnScanReport
-------------	----------------------------

Remark The distance between the start point projected to the cylinder axis and the start point (S_x, S_y, S_z) may not be zero and defines the diameter of the cylinder.

During the scan the tool center will move within the surface (ScanningCylinder), that is created by rotating a line ($S_x, S_y, S_z, C_i, C_j, C_k$) around the cylinder axis.

The distance between the start point (S_x, S_y, S_z) and the direction point (D_x, D_y, D_z) may not be zero.

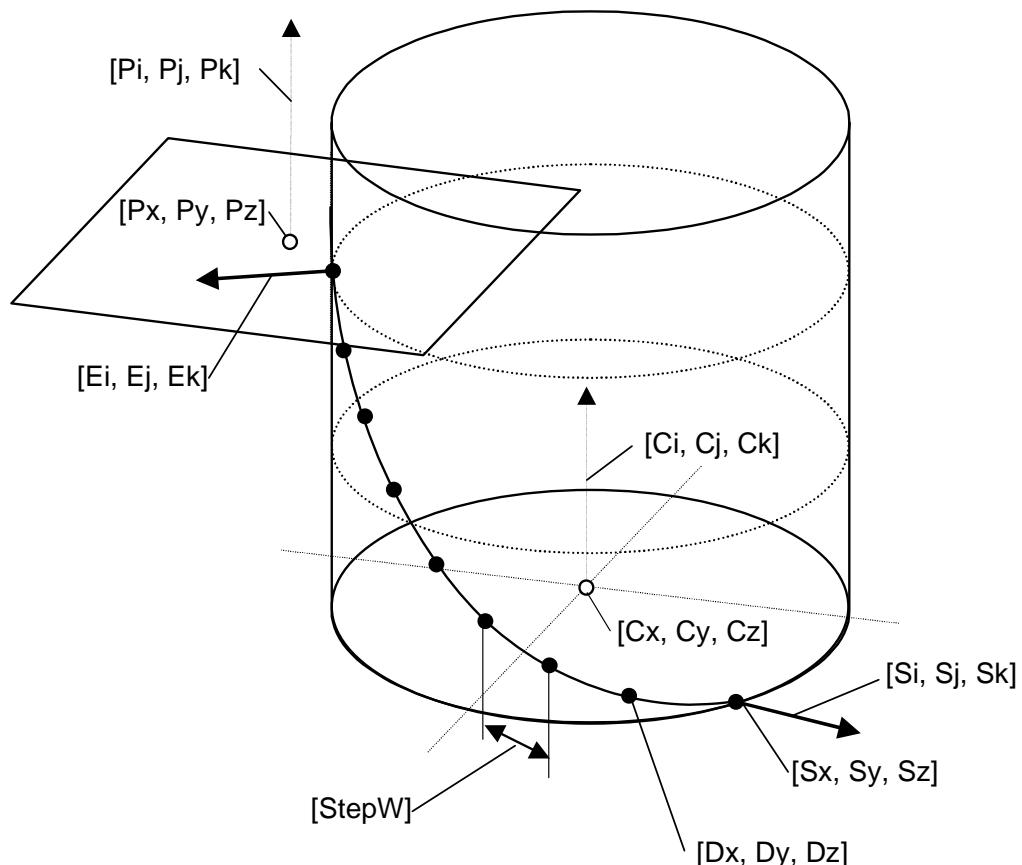
The scan is executed as follows:

The DME will implicitly execute a PtMeas command using (S_x, S_y, S_z) as point and (S_i, S_j, S_k) as surface normal.

The DME will start to scan into the direction from start to direction point.
During the scan the tool center will move ScanningCylinder.

The DME will stop scanning when it passes n times through the stop plane.

The DME will start to check the stop criteria when it moved a distance that is larger than the distance between start and direction point.



11.4 Scanning Examples

11.4.1 Scanning known contour circle

Client to Server	Server to Client	Comment
00014 OnScanReport(X(),Y(),Z(),Q())		Client defines format for scanning result. Valid for every scanning command from now on.
	00014 &	
	00014 %	
00015 ScanOnCircleHint (0.01, 0,001)		Gives as a hint prognosted Displacement and Form
	00015 &	
	00015 %	
00016 ScanOnCircle (100, 0, -3, 120, 0, -3, 0, 0, 1, 360, 180, 0.5)		Indizes are: ScanOnCircle(Cx, Cy, Cz, Sx, Sy, Sz, I, j, k, delta, sfa, StepW)
	00016 &	
	00016 # 118.5, 0.0001, - 3.0002, 0	Scanning result from server, one point, assuming probe sphere radius is 1.5mm
	00016 # 118.4992,0,1614,3.0002, 0,118.4971,0.3228,3.0002 ,100,	Multiple scanning results points blocked in one result string
	Follow multiple times until all scanning results are transmitted
	00016 %	Scanning ready

11.4.2 Scanning unknown contour

Client to Server	Server to Client	Comment
		Previous defined OnScanReport is used
00015 ScanUnknownHint (100.0)		Gives as a hint prognosted minimum radius of curve
	00015 &	
	00015 %	
00016 ScanInPlaneEndIsSphere		Indizes are:

(100,0,0,0,0,1,100,1,0,0.2,100,1 00,1.5,1.0,0,0,1)		ScanInPlaneEndIsSphere(Sx,S y,Sz,Si,Sj,Sk,Dx,Dy,Dz,Step W,Ex,Ey,Ez,Dia,Ei,Ej,Ek)
	00016 &	
	00016 # 100.0000, 0.0000,1.5000,0,100.0001 ,0.2000,1.5000,0,100.000 0,0.4000,1.5000,0	Scanning result from server, three points
	Multiple scanning results
	00016 # 100.0000,99.8000,1.5000, 0,100.0000,100.0000,1.50 00,0	Follow multiple times until end criteria is reached and all scanning results are transmitted
	00016 %	Scanning ready

12 Rotary Table

12.1 AlignPart()

The client uses this method to force the part to be orientated according to the given vector(s).

- AlignPart(px1, py1, pz1, mx1, my1, mz1, alpha)
- AlignPartl(px1, py1, pz1, mx1, my1, mz1,
px2, py2, pz2, mx2, my2, mz2, alpha, beta)

Parameters First command for single rotary tables.

Two normalized vectors (px, py, pz, mx, my, mz). The first vector is in part coordinates. The second vector is in machine coordinates.

Maximal allowed error angle (alpha) in which the found orientation may differ from the desired one projected to the rotation plane. In case the angle exceeds, “Part not aligned” is returned. In case alpha is zero no error check is performed.

Second command if applicable when two rotational equipments rectangular to each other are available.

Data Returns vectors (same number as set) which describe the reached alignment.

Errors 2506: Part not aligned.

In case of a rotary table both vectors are projected in the pane of rotation. After projection both vectors must be normalizable.

Appendix A C++ and Header Files for Explanation

A.1 \main\main.cpp

```
-----  
#include "../cartcmm/cartcmm.h"  
//#include "../dme/dme.h"  
//#include "../cartcmmwithrottbl/CartCmmWithRotTbl.h"  
  
-----  
//Server _Server;  
//Server* Srv() {return &_Server;}  
  
void main () {  
  
    //r8 speed = Srv()->GoToPar()->Speed();  
    //ie r = Srv()->GoToPar()->Speed(5.0);  
  
    -----  
};
```

A.2 \server

A.2.1 \server\server.h

```
#if !defined(AFX_Server_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)  
#define AFX_Server_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_  
  
#include "Part.h"  
#include <ETag.h>  
  
-----  
class Server {  
  
    Part _Part;  
  
-----  
public: Server ();  
virtual ~Server ();  
  
-----  
void StartSession (cTag tag); // connect to client  
void EndSession (cTag tag); // disconnect from client  
ie StopDaemon (cTag tag, cETag &fqt); // stop daemon  
ie StopAllDaemons (cTag tag); // stop all daemons  
void AbortE (cTag tag); // abort pending transactions  
void GetErrorInfo (cTag tag); // abort pending transactions  
void ClearAllErrors (cTag tag);  
  
-----  
virtual void GetProp (cTag tag, ...);  
virtual void GetPropE (cTag tag, ...);  
virtual void SetProp (cTag tag, ...);  
virtual void EnumProp (cTag tag, ...);  
virtual void EnumAllProp (cTag tag, ...);  
  
-----  
private:  
    // these methods are for  
    // documentation purpose only  
  
    void MainLoop ();  
    void DispatchToEventQue (Tag *tag, String &command);  
    void Dispatch (Tag *tag, String &command);  
    void SendAck (Tag *tag);  
    void SendData (Tag *tag, cString &response);
```

```

void      SendError          (Tag *tag, cErrorSeverity sev, cErrorCode code);
void      SendReady          (Tag *tag);
void      FormatTag          (String &response, Tag* tag);
i4       DecodeTag           (cString &command);
void      Transmit           (cString &response);

//-----
bool      ServerIsAlive();
ErrorSeverity GetErrorSeverity();
ErrorCode    GetErrorCode();
bool       ErrorDuringCommandExecution();

//-----
};

#endif

```

A.2.2 \server\part.h

```

#ifndef !defined(AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_Part_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include <IppTop.h>

//-----

class Part {

//-----

r8      _Approach;
r8      _XpanCoefficient;
r8      _Temperature;

//-----

public:   Part      ();
virtual ~Part     ();

//-----

r8      Approach()      {return _Approach;}

//-----
};

#endif

```

A.2.3 \server\server.cpp

```

//-----

#include "String.h"
#include "Server.h"

//-----

void    Server::MainLoop()                                { // for documentation only ***

    // this method is implemented for

    // documentation purpose only
String   command;
Tag*    tag      = Nil;
do {
    // wait for a command line from client
    // .. Wait(command)

i4
from 2 to 5

    tagval = DecodeTag(command);           // get tag values define by chars

    if (command.FirstCharIs('E')) {
        tag = new ETag(tagval);

```

```

        SendAck(tag);                                // 
confirmme receive
        DispatchToEventQue(tag, command);} // do whatever is necessary

        else {
            tag = new Tag(tagval);
            SendAck(tag);                      // confirmme receive
            Dispatch(tag, command);}           // do whatever is necessary

        if (ErrorDuringCommandExecution()) {    // something went wrong ?
            SendError(tag, GetErrorSeverity(), GetErrorCode());}      // send error
message

        SendReady(tag);}
        while (ServerIsAlive());                // while server is alive
//-----

void Server::Transmit(cString &response) {           // for documentation only ***

    // send this string to client

    /*... Send(response) */}

//-----

void Server::FormatTag(String &response, Tag* tag){ // for documentation only ***
remove all chars
    response.SetLen0();                         //
zeros
    response.Format(tag->Val(), 5);           // format 5 digits with leading
if (dynamic_cast<ETag*>(tag) !=Nil) {
    response[1] = 'E';}                      // use E to indicate event
    response += " ";}                         // append a space char
//-----

void Server::SendAck(Tag* tag) {           // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "&";                         // add %
    Transmit(response);}

//-----

void Server::SendData(Tag *tag, cString &data) {       // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "# ";
    response += data;                        // add # and space
    Transmit(response);}

//-----

void Server::SendError(Tag *tag, cErrorSeverity sev, cErrorCode code) {      // for
documentation only ***

String      response;
    FormatTag(response, tag);
    response += "! ";
    response += ...;                      // add ! and space
//      response += ...;                  // add error
    Transmit(response);}

//-----

void Server::SendReady(Tag *tag) {           // for documentation only ***
String      response;
    FormatTag(response, tag);
    response += "%";                     // add %
    Transmit(response);}

//-----

```

A.3 \dme

A.3.1 \dem\dme.h

```
#if !defined(AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_DME_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../server/Server.h"
#include "../toolchanger/ToolChanger.h"

//-----
class DME : public Server {

    ToolChanger          _ToolChanger;
//-----

    bool                  _IsHomed;
    bool                  _IsUserEnabled;
//-----

public:           DME      ();
virtual         ~DME () ;
//-----

    ToolChanger* TCh()                                {return &_ToolChanger; }

//-----

    virtual ie          Home                           (cTag tag)
    {
        i4           IsHomed                         (cTag tag)
        {return _IsHomed; }
    }

//-----

    virtual void   EnableUser                        (cTag tag)      {}
    virtual void   DisableUser                       (cTag tag)      {}
    bool          IsUserEnabled                     (cTag tag)      {}

//-----

    ie            OnPtMeasReport                    (cTag tag, ...);
    ie            OnMoveReportE                   (cETag tag, cr8 dis, cr8 time,...);

//-----

    void          GetMachineClass                  (cTag tag)
    void          GetErrorStatusE                 (cTag tag)
    void          GetXtdErrorStatus              (cTag tag)
    {}

//-----

    void          Get                           (cTag tag, ...);
    ie            GoTo                          (cTag tag,...);
    ie            PtMeas                        (cTag tag,...);
    ie            PtMeasIJK                     (cTag tag,...);
    KTool*       Tool                          () const {return _ToolChanger._ActTool;}
    ie            FindTool                      (cTag tag, cString &name) {return TCh()->FindTool (tag,
    name);}
    KTool*       FoundTool                     () const {return _ToolChanger._FoundTool;}
    ie            ChangeTool                    (cTag tag, cString &name) {return TCh()->ChangeTool (tag,
    name);}
    ie            SetTool                       (cTag tag, cString &name) {return TCh()->SetTool   (tag,
    name);}
    ie            AlignTool                     (cTag tag, cv3 &ijk, cr8 alpha) {return Tool()->AlignTool
    (tag, ijk, alpha);}

}
```

```

ie      AlignTool      (cTag tag, cV3 &ijk, cV3 &uvw, cr8 alpha, cr8 beta)
{return Tool()->AlignTool (tag, ijk, uvw, alpha, beta);}
GoToPars* GoToPar      ()      const {return Tool()->GoToPar();}
PtMeasPars* PtMeasPar   ()      const {return Tool()->PtMeasPar();}
GoToPars* ABCGoToPar   ()      const {return Tool()->ABCGoToPar();}
PtMeasPars* ABCPtMeasPar ()      const {return Tool()->ABCPtMeasPar();}

//-----

virtual r8    X      () ;           // return machine position in the
virtual r8    Y      () ;           // selected coordinate system
virtual r8    Z      () ;
virtual V3    IJK    () ;
virtual ie    X      (cr8 x);      // move machine to target position
virtual ie    Y      (cr8 y);
virtual ie    Z      (cr8 z);
virtual ie    IJK    (const V3 &ijk);

//-----

ie          OnScanReport      (cTag tag, ...);
ie          ScanOnCircleHint  (cTag tag, ...);
ie          ScanOnCircle      (cTag tag, ...);
ie          ScanOnLineHint    (cTag tag, ...);
ie          ScanOnLine        (cTag tag, ...);

ie          ScanUnKnownHint   (cTag tag, ...);
ie          ScanInPlaneEndIsSphere(cTag tag, ...);
ie          ScanInPlaneEndIsPlane (cTag tag, ...);
ie          ScanInPlaneEndIsCyl  (cTag tag, ...);
ie          ScanInCylEndIsSphere (cTag tag, ...);
ie          ScanInCylEndIsPlane  (cTag tag, ...);

};//-----
#endif

```

A.4 \cartcmm

A.4.1 \cartcmm\cartcmm.h

```

#ifndef !defined(AFX_CartCMM_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_CartCMM_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../DME/dme.h"
#include "T33.h"

//-----

class CartCMM :public DME {

Axis      _XAxis;
Axis      _YAxis;
Axis      _ZAxis;

//-----

CoordSys  _CoordSys;
T33       _PartCoordTrandformation;

//-----

public:    CartCMM ();
virtual ~CartCMM      ();

//-----

Axis*     XAx()  {return &_XAxis;}
Axis*     YAx()  {return &_YAxis;}
Axis*     ZAx()  {return &_ZAxis;}

//-----

virtual

```

```

ie      SetCoordSystem(CoordSys csy);
CoordSys GetCoordSystem()           {return _CoordSys;}

//-----

ie      SetCsyTransformation(const T33EA &tra);
T33EA  GetCsyTransformation();

//-----


protected:

virtual r8      X();                                // return machine position in the
virtual r8      Y();                                // selected coordinate system
virtual r8      Z();
virtual V3      IJK();

virtual ie      X(cr8 x);                           // move machine to target position
virtual ie      Y(cr8 y);
virtual ie      Z(cr8 z);
virtual ie      IJK(const V3 &iijk);

//-----
};

#endif

```

A.4.2 \cartcmm\ eulerw.cpp

```

// EulerA.cpp: implementation of the EulerA class.

#include "R33.h"
#include "EulerW.h"

//-----


cr8      R_Delta = 1e-12;

r8      abs  (cr8 x);
r8      sind (cr8 x);
r8      cosd (cr8 x);
r8      Acosd (cr8 x);
r8      Atan2d(cr8 y, cr8 x);

//-----


EulerA::EulerA(cR33 &b){                                     // create Euler from
    _Psi  = 0,
    // rotation matrix
    _Phi  = 0;
r8      s3   = 0,
r8      c3   = 0,
r8      c1   = b.Val(3,3);
    _Theta = Acosd(c1);
r8      s1   = sind(_Theta);
    if (abs(s1) > R_Delta) {                                // check if Thet() is 0
        s2   = b.Val(1,3)/s1;                               // no calculate Psi()
        c2   = -b.Val(2,3)/s1;
        _Psi = Atan2d(s2, c2);
    }
    EulerA      ew(_Theta, _Psi, _Phi);                      // use Thet(), Psi(), 0 to create matrix
R33      r(ew); -r;
    // and calclate Psi() from orig mat
    rr(b); rr*=r;                                         // and matriz build from Thet() Psi()
    c3   = r.Val(1,1);
    s3   = r.Val(2,1);}
    else {
        // Thet()=0, Psi()==0
        c3   = b.Val(1,1);                                // calculate phi
        s3   = b.Val(2,1);}
        _Phi = Atan2d(s3, c3);}

//-----


void    R33::Create(cEulerA &b) {

r8      c1      = cosd(b.Thet()),          // theta==0 && psi==0
        s1      = sind(b.Thet()),          // c3      s3      0
        c2      = cosd(b.Psi()),          // c3      s3      0

```

```

s2          = sind(b.Psi()),      //      -s3      c3      0
c3          = cosd(b.Phi()),     //      0       0       1
s3          = sind(b.Phi());

Mat(1,1) = c2*c3-c1*s2*s3;
Mat(1,2) = s2*c3+c1*c2*s3;
Mat(1,3) = s1*s3;

Mat(2,1) = -c2*s3-c1*s2*c3;
Mat(2,2) = -s2*s3+c1*c2*c3;
Mat(2,3) = s1*c3;
//      c2*c3-c1*s2*s3      s2*c3+c1*c2*s3      s1*s3

Mat(3,1) = s1*s2;
//      -c2*s3-c1*s2*c3      -s2*s3+c1*c2*c3      s1*c3
Mat(3,2) = -s1*c2;
Mat(3,3) = c1;
//      s1*s2      -s1*c2      c1

//-----

```

A.5 \cartcmmwithrottbl

A.5.1 \cartcmmwithrottbl\cartcmmwithrottbl.h

```

#ifndef !defined(AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_)
#define AFX_CartCmmWithRotTbl_H__E4F9759D_0A8F_11D3_A3F2_0000F87ABD00__INCLUDED_

#include "../cartcmm/cartcmm.h"

//-----

class CartCmmWithRotTbl: public CartCMM {

//-----

Axis           _RAxis;

//-----

public:          CartCmmWithRotTbl      ();
virtual         ~CartCmmWithRotTbl    ();

//-----

Axis*           RAx      () {return &_RAxis;}

//-----

ie             AlignPart      (cTag      tag, ...);

//-----

virtual char*   Type      () {return "CartCMMWithRotTbl";}

};//-----
#endif

```

A.6 \toolchanger

A.6.1 \toolchanger\toolchanger.h

```

// ToolChanger.h: interface for the ToolChanger class.

#ifndef !defined(AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112__INCLUDED_)
#define AFX_ToolChanger_H__2418E2DB_F44D_4F25_B290_3EDC4854E112__INCLUDED_

#include "ToolAB.h"

//-----

class ToolChanger {                                     friend class DME;

```

```

KTool*          _ActTool;
KTool*          _FoundTool;
KTool*          _DefaultTool;
KTool*          _UndefTool;

//-----

Ary<KTool*>    _Tools;
V3              _TransferPosition;

//-----


ToolChanger     (){

/*
String      name      = "DefaultTool"
            _DefaultTool = new Tool(name);

            name      = "UndefTool"
            _UndefTool = new Tool(name);

            name      = "NoTool"
Tool*       tool      = new Tool(name);
            _Tool.Add(tool);

            name      = "ReferenceTool"
Tool*       tool      = new Tool(name);
            _Tool.Add(tool);

*/
}

//-----


virtual        ~ToolChanger();

//-----


KTool*          ActTool()      const {return _ActTool;}
KTool*          FoundTool()    const {return _FoundTool;}

//-----


GoToPars*        GoToPar()      const {GoToPars* r=_ActTool()->GoToPar ();   if (r==Nil)
r=_DefaultTool->GoToPar(); return r;}
GoToPars*        ABCGoToPar()  const {GoToPars* r=_ActTool()->ABCGoToPar(); if (r==Nil)
r=_DefaultTool->ABCGoToPar(); return r;}

//-----


PtMeasPars*      PtMeasPar()   const {PtMeasPars* r=_ActTool()->PtMeasPar ();   if
(r==Nil) r=_DefaultTool->PtMeasPar(); return r;}
PtMeasPars*      ABCPtMeasPar() const {PtMeasPars* r=_ActTool()->ABCPtMeasPar(); if
(r==Nil) r=_DefaultTool->ABCPtMeasPar(); return r;}

//-----


i4              Howmany(cTag tag)   {return _Tools.Len();}

//-----


ie              Qualify   (cTag tag)           {return _ActTool->Qualify(tag);}
ie              ChangeTool (cTag tag, cString &name);
ie              SetTool    (cTag tag, cString &name);
ie              FindTool   (cTag tag, cString &name){_FoundTool = Find(tag, name); return
(_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
ie              FindTool   (cTag tag, CV3     &ijk)  {_FoundTool = Find(tag, ijk); return
(_FoundTool==Nil) ? ErrorToolNotFound : ErrorSuccess;}
String          ActToolName(cTag tag)        {return _ActTool->Name();}

//-----


void            EnumTools(cTag tag) {
String          name;
for (i4 i=0; i < _Tools.Len(); ) {
    name = _Tools[i]->Name();
    /*send name to client*/}}

```

```

//-----
private:
KTool*           Find(cTag tag, cString &name) /* for(i..) toolname = _Tools[i]-
>Name() */;
KTool*           Find(cTag tag, cV3      &ijk) /* for(i..) toolname = _Tools[i]-
>Name() */;

//-----
};

#endif

```

A.6.2 \toolchanger\tool.h

```

// Tool.h: interface for the Tool class.

#ifndef AFX_Tool_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
#define AFX_Tool_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "String.h"
#include "GoToParams.h"
#include "PtMeasPars.h"
#include "V3.h"
#include "Axis.h"

//-----

class KTool  {
friend class ToolChanger;

String          _Name;
i4              _Type;

//-----

GoToPars*        _GoToPar;
GoToPars*        _ABCGoToPar;

//-----

PtMeasPars*      _PtMeasPar;
PtMeasPars*      _ABCPtMeasPar;

//-----

String          _QualificationArtifact;
i4              _QualificationState;
DateTime        _LastQualificationDate;
ui              _MethodsSupported;

//-----

public:
    KTool(cString &name);
    virtual ~KTool();

//-----

String          Name()          {return _Name;}

//-----

GoToPars*        GoToPar      () const {return _GoToPar;}
GoToPars*        ABCGoToPar() const {return _ABCGoToPar;}

//-----

PtMeasPars*      PtMeasPar() const {return _PtMeasPar;}
PtMeasPars*      ABCPtMeasPar() const {return _ABCPtMeasPar;}

//-----

bool            CanDoGoTo  ();
bool            CanDoPtMeas();

//-----

ie              Qualify(cTag tag);

//-----

```

```

virtual ie      Align    (cTag tag, cv3 &ijk)
               {return ErrorBadContext;}
virtual ie      AlignTool (cTag tag, cv3 &ijk, cr8 alpha)
               {return ErrorBadContext;}
virtual ie      AlignTool (cTag tag, cv3 &ijk, cv3 &uvw, cr8 alpha, cr8 beta)
               {return ErrorBadContext;}

//-----

virtual void   EnumProp   (cTag     tag, ...);
virtual void   GetProp    (cTag     tag, ...);
virtual void   GetPropE   (cTag     tag, ...);
virtual void   SetProp    (cTag     tag, ...);

//-----

protected:
virtual r8      A()                                {return 0;}
virtual r8      B()                                {return 0;}
virtual r8      C()                                {return 0;}

virtual ie      A(cr8 a)                           {return ErrorSuccess;}
virtual ie      B(cr8 b)                           {return ErrorSuccess;}
virtual ie      C(cr8 c)                           {return ErrorSuccess;}

//-----
};

#endif

```

A.6.3 \toolchanger\toolab.h

```

// ToolAB.h: interface for the ToolAB class.

#ifndef _AFX_TOOLAB_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
#define _AFX_TOOLAB_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "Tool.h"

//-----

class ToolAB : public KTool {
public:
    Axis          _AAxis;
    Axis          _BAxis;

//-----

public:
    ToolAB(cString &name);
    ~ToolAB();

//-----

ie      Align    (cTag tag, cv3 &ijk);
void   EnumProp(cTag tag);

//-----

r8      A();
r8      B();

ie      A(cr8 a);
ie      B(cr8 b);

//-----

};

#endif

```

A.6.4 \toolchanger\toolabc.h

```
// ToolABC.h: interface for the ToolABC class.
```

```

#ifndef _AFX_ToolABC_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
#define _AFX_ToolABC_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "ToolAB.h"

//-----
class ToolABC : public ToolAB {
    Axis           _CAxis;
//-----

public:          ToolABC(const String &name);
virtual         ~ToolABC();

//-----

    ie          Align  (cTag tag, cV3 &ijk);
    void        EnumProp(cTag tag);

//-----

    r8          C();
    ie          C(cr8 c);

//-----
};

#endif

```

A.6.5 \toolchanger\gotoparams.h

```

// GoToPars.h: interface for the GoToPars class.

#ifndef _AFX_GoToPars_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_
#define _AFX_GoToPars_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "../lib/String.h"
#include "Param.h"

//-----

class GoToPars {

    Param      _Speed;
    Param      _Accel;

//-----

public:        GoToPars();
virtual        ~GoToPars();

//-----

    r8          MinSpeed()           const {return _Speed.Min();}
    r8          Speed()             const {return _Speed.Val();}
    r8          MaxSpeed()          const {return _Speed.Max();}
    bool        CanChangeSpeed()    const {return _Speed.CanChange();}
    ie          Speed(cr8 s)       {return _Speed.Val(s);}

//-----

    r8          MinAccel()          const {return _Accel.Min();}
    r8          Accel()             const {return _Accel.Val();}
    r8          MaxAccel()          const {return _Accel.Max();}
    bool        CanChangeAccel()   const {return _Accel.CanChange();}
    ie          Accel(cr8 s)       {return _Accel.Val(s);}

//-----

    void        EnumProp();

//-----
};

#endif

```

A.6.6 \toolchanger\ptmeaspars.h

```
// PtMeasPars.h: interface for the PtMeasPars class.

#ifndef !defined(AFX_PtMeasPars_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_)
#define AFX_PtMeasPars_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "../lib/String.h"
#include "../lib/DateTime.h"
#include "GoToParams.h"

//-----

class PtMeasPars  {

Param      _Approach;
r8         _Search;
r8         _Retract;
GoToPars   _Move;

//-----

public:     PtMeasPars();
virtual    ~PtMeasPars();

//-----

r8         MinSpeed()          {return _Move.MinSpeed();}
r8         Speed   ()          {return _Move.Speed();}
r8         MaxSpeed()          {return _Move.MaxSpeed();}
ie        Speed   (cr8 s)     {return _Move.Speed(s);}

//-----

r8         MinAccel()          {return _Move.MinAccel();}
r8         Accel   ()          {return _Move.Accel();}
r8         MaxAccel()          {return _Move.MaxAccel();}
ie        Accel   (cr8 s)     {return _Move.Accel(s);}

//-----

void       EnumProp();

//-----
};

#endif
```

A.6.7 \toolchanger\param.h

```
// Param.h: interface for the Param class.

#ifndef !defined(AFX_Param_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_)
#define AFX_Param_H__79AF9D2B_A7BC_4F04_923D_1452AF559CC1__INCLUDED_

#include "IppTypeDef.h"
#include <IppErrorCodes.h>

r8           min(cr8 a, cr8 b);
r8           max(cr8 a, cr8 b);

//-----

class Param  {

r8           _Min;
r8           _Val;
r8           _Max;
bool        _CanChange;

//-----

public:     Param() {_Min=-10000; _Val=0; _Max=10000; _CanChange=Tr;}
virtual    ~Param();

//-----
```

```

r8      Min      ()          const {return _Min;}
r8      Val      ()          const {return _Val;}
r8      Max      ()          const {return _Max;}
bool    CanChange()        const {return _CanChange || (_Max-_Min)<=0;}
void    EnumProp();
```

```

ie      Val(cr8 v)           {
ie      errcod = ErrorSuccess;
bool   r      = CanChange();
      if (r) {
        r = v >= _Min && v <= _Max;
        if (r) {
          _Val=v;
        }
        else {
          _Val  = min(v,      _Max);
          _Val  = max(_Val, _Min);
          errcod = (v < _Min) ? ErrorParamTooSmall : ErrorParamTooLarge;}}
      else {
        errcod = ErrorParamCannotBeChanged;
      }
      return errcod;
```

```

private:
void    Min      (cr8 v) { _Min=v; }
void    Max      (cr8 v) { _Max=v; }
void    CanChange(cbo v) { _CanChange=v; }
```

```

};
```

```
#endif
```

A.7 Most important of lib

A.7.1 \lib\axis.h

```

// Axis.h: interface for the Axis class.

#ifndef AFX_Axis_H__B3DA30C7_5415_11D3_A481_0000F87ABD00__INCLUDED_
#define AFX_Axis_H__B3DA30C7_5415_11D3_A481_0000F87ABD00__INCLUDED_

#include "IppTypeDef.h"

class Axis {

enum    AxisType      {Lin=1, Rot=2};

char    _Name[8];
AxisType _Type;
r8     _MinPos;
r8     _ActPos;
r8     _MaxPos;
r8     _Pitch;
r8     _Temperature;
bool   _IsControlled;
bool   _IsHomed;

public:    Axis();
virtual ~Axis(){};
```

```

i4      Type()           const {return _Type;}
r8      MinPos()         const {return _MinPos;}
r8      MaxPos()         const {return _MaxPos;}
r8      Pitch()          const {return _MaxPos;}
r8      Temperature()    const {return _Temperature;}
```

```

static void EnumProp(); // Name, c*8
                      // Type, i4
                      // MinPos, r8
                      // MaxPos, r8
                      // Temperature, r8
//-----
};

#endif

```

A.7.2 \lib\ eulerw.h

```

// EulerA.h: interface for the EulerA class.

#if !defined(AFX_EulerA_H__0E096DA3_5537_11D3_84A8_0000F87ADB6B__INCLUDED_)
#define AFX_EulerA_H__0E096DA3_5537_11D3_84A8_0000F87ADB6B__INCLUDED_

#include "IppTop.h"

//-----

class EulerA {
    r8 _Theta; // Euler angel in degree
    r8 _Psi;
    r8 _Phi;

//-----

public: EulerA();
        EulerA(cr8 theta, cr8 psi, cr8 phi);
        EulerA(cR33 &b);
    virtual ~EulerA();
//-----

    r8 Tht() const {return _Theta;}
    r8 Psi() const {return _Psi;}
    r8 Phi() const {return _Phi;}
//-----

};

#endif

```

A.7.3 \lib\ tag.h

```

// KTag.h: interface for the KTag class.

#if !defined(AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_)
#define AFX_KTag_H__001F2611_6298_11D3_A49B_0000F87ABD00__INCLUDED_

#include <IppTypeDef.h>

//-----

class Tag {
    static i4 _TagCounter; // static tag
    counter i4 _Tag; // tag

//-----

public: Tag(ci4 i) {_Tag = i;}
    virtual ~Tag() {}

//-----

    i4 Val() {return _Tag;}
    i4 NewTag(); // create new tag *** for client use only
//-----

```

```
};  
#endif
```

A.7.4 \lib\ippstypedef.h

```
// This is the global type definition file  
  
#ifndef _IppTypeDefDefined  
#define _IppTypeDefDefined  
  
//-----  
  
typedef      unsigned      char      uc;          // define some shortcuts  
typedef const unsigned      char      cuc;         // for type definitions  
  
typedef      unsigned      char      ch;  
typedef const unsigned      char      cc;  
  
typedef      unsigned      short     ui2;  
typedef      signed       short     i2;  
typedef const signed       short     ci2;  
typedef const unsigned      short     cui2;  
  
typedef      signed       int      i4;  
typedef const signed       int      ci4;  
  
typedef      signed       int      ie;           // error codes  
typedef const signed       int      cie;  
  
typedef const unsigned      int      cui;  
typedef      unsigned      int      ui;  
  
typedef      unsigned      int      ich;  
typedef const unsigned      int      cich;  
  
typedef      double       r8;  
typedef const double       cr8;  
typedef      float        r4;  
typedef const float        cr4;  
  
typedef      const bool    bo;  
typedef      const bool    cbo;  
  
//-----  
  
#define      Fa      false   // define boolean shortcuts  
#define      Tr      true  
#define      Nil     0  
  
//-----  
#endif
```

A.7.5 \lib\ippbaseclasses.h

```
// predefined classes  
  
#ifndef _IppBaseClassesDefined  
#define _IppBaseClassesDefined  
  
//-----  
  
class String;      typedef const String      cString;      // data base object  
class Tag;        typedef const Tag        cTag;        // data base object  
class ETag;       typedef const ETag       cETag;       // data base object  
class GoToPars;   typedef const GoToPars   cGoToPars;   // data base object  
class PtMeasPars; typedef const PtMeasPars cPtMeasPars; // data base object  
  
//-----  
  
class V3;          typedef const V3        cV3;        // data base object  
class M33;         typedef const M33       cM33;       // data base object  
class R33;         typedef const R33       cR33;       // data base object  
class T33;         typedef const T33       cT33;       // data base object
```

```
class T33EA;           typedef const T33EA      cT33EA;      // data base object
class EulerA;          typedef const EulerA   cEulerA;     // data base object
//-----
class Axis;            typedef const Axis       cAxis;
class Part;             typedef const Part      cPart;
//-----
enum ErrorSeverity;    typedef const ErrorSeverity cErrorSeverity;
enum ErrorCode;         typedef const ErrorCode   cErrorCode;
//-----
#endif
```